Transitive Closure Logic and Multihead Automata with Nested Pebbles

Minna Hirvonen

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta/Osasto — Fakultet/Sektion — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Mathematics and Statistics

Tekijä — Författare — Author
Minna Hirvonen

Työn nimi — Arbetets titel — Title

Transitive Closure Logic and Multihead Automata with Nested Pebbles

Oppiaine — Läroämne — Subject
Mathematics

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's thesis	May 2020	43

Tiivistelmä — Referat — Abstract

Several extensions of first-order logic are studied in descriptive complexity theory. These extensions include transitive closure logic and deterministic transitive closure logic, which extend first-order logic with transitive closure operators. It is known that deterministic transitive closure logic captures the complexity class of the languages that are decidable by some deterministic Turing machine using a logarithmic amount of memory space. An analogous result holds for transitive closure logic and nondeterministic Turing machines.

This thesis concerns the k-ary fragments of these two logics. In each k-ary fragment, the arities of transitive closure operators appearing in formulas are restricted to a nonzero natural number k. The expressivity of these fragments can be studied in terms of multihead finite automata. The type of automaton that we consider in this thesis is a two-way multihead automaton with nested pebbles.

We look at the expressive power of multihead automata and the k-ary fragments of transitive closure logics in the class of finite structures called word models. We show that deterministic two-way k-head automata with nested pebbles have the same expressive power as first-order logic with k-ary deterministic transitive closure. For a corresponding result in the case of nondeterministic automata, we restrict to the positive fragment of k-ary transitive closure logic. The two theorems and their proofs are based on the article 'Automata with nested pebbles capture first-order logic with transitive closure' by Joost Engelfriet and Hendrik Jan Hoogeboom. In the article, the results are proved in the case of trees. Since word models can be viewed as a special type of trees, the theorems considered in this thesis are a special case of a more general result.

Avainsanat — Nyckelord — Keywords
First-order logic, transitive closure, multihead automata, pebbles, word models

Säilytyspaikka — Förvaringsställe — Where deposited
Kumpula Campus Library

Muita tietoja — Övriga uppgifter — Additional information

Contents

Chapter 1

Introduction

The most prominent logic studied in classical model theory is first-order logic FO. As a consequence of the compactness theorem, any first-order axiom system without infinite models can only have finite models of limited cardinality. Since finitely many finite structures can be described explicitly by a single first-order sentence, classical model theory is mostly concerned with infinite structures.

However, this does not mean that finite structures are uninteresting. Finite structures appear naturally in many contexts, especially in questions about computation. The subarea of model theory where finite structures are studied is called finite model theory. Many central theorems of classical model theory do not hold when we are restricted to finite structures. For this reason, methods of proof are quite different in the finite setting. One important result in finite model theory is Trahtenbrot's theorem (1950) that demonstrates the failure of the completeness theorem in the finite. The proof of the theorem (see e.g. [4]) relies on the undecidability of the halting problem, and thus it is an example of how finite model theory is linked to (theoretical) computer science.

One of the main areas of application for finite model theory is descriptive complexity. Descriptive complexity theory is an area of computational complexity theory that characterizes complexity classes by determining a corresponding logic for them: every class of finite (ordered) structures, a *language*, definable in the logic has to be in the complexity class, and the logic has to have enough expressive power for defining any such language in the complexity class. If a logic corresponds to the complexity class in this way, the logic is said to *capture* that complexity class.

Since first-order logic is not expressive enough for some features of computation, several of its extensions are studied in the field of descriptive complexity. *Transitive closure logic* FO(TC) and *deterministic transitive closure logic* FO(DTC), both of which extend first-order logic with transitive closure operators, are examples of such extensions. In 1987, Immerman [10] showed that logics FO(DTC) and FO(TC) capture the space com-

plexity classes DSPACE($\log n$) and NSPACE($\log n$), respectively. The class DSPACE($\log n$) consists of the languages that are decidable by some deterministic Turing machine using a logarithmic amount of memory space. The class NSPACE($\log n$) is defined analogously, but the Turing machines are allowed to be nondeterministic. It is an open problem in complexity theory whether these two classes are the same.

Besides Turing machines, there are also other mathematical models of computation. One example of these are finite automata for which the usage of memory is more limited than for Turing machines. Finite automata were the subject of much research in the 1940's and 1950's, and they were originally considered for modelling brain function – or more precisely neural networks. Later, they have shown to be useful for a variety of other purposes, particularly in computer science [8]. They are also studied in finite model theory. In fact, the historically first logical characterization of a complexity class was the characterization of the class of languages accepted by finite automata by means of monadic second-order logic (see e.g. [4]). The result was obtained in the 1960's independently by Büchi, Elgot, and Trakhtenbrot. By now, many different types of finite automata have been introduced, for example, multihead automata which have more than one head for reading the input, and pebble automata with a finite set of pebbles that are used for marking positions on the input tape.

In this thesis, we consider the fragments FO(DTC^k) and FO(TC^k) of transitive closure logics. In these fragments, the arities of transitive closure operators appearing in formulas are restricted to k, where $k \geq 1$. Additionally, in the case of transitive closure logic FO(TC), where transitive closure operators can be nondeterministic, we further restrict to the fragments FO(posTC^k) containing only positive occurrences of transitive closure.

We study the expressive power of these fragments in the class of finite structures called word models in terms of multihead finite automata. The automaton model that we work with is a *two-way multihead automaton with nested pebbles*. It is a finite automaton that has k heads for reading the input. The automaton can move these heads both left and right on the input tape, and mark specific positions on the tape by using pebbles in nested fashion.

The main theorem in this thesis concerns deterministic k-head automata and fragments FO(DTC^k) in word models. The theorem was proved more generally for trees in the article *Automata with nested pebbles capture first-order logic with transitive closure* by Joost Engelfriet and Hendrik Jan Hoogeboom [5]. Based on the proofs presented in the article, we show that deterministic two-way k-head automata with nested pebbles capture first-order logic with k-ary deterministic transitive closure. In the article, a nondeterministic version of the theorem is proved for the fragments FO(posTC^k). The proofs for deterministic and nondeterministic versions are similar, so only the necessary modifications to the proof for the nondeterministic version are covered.

The thesis is divided into seven chapters. We give most of the necessary definitions in Chapters 2 and 3. In these chapters, we define transitive closure logics and word models, and give some examples to help the reader understand or recall the concepts. We also consider the automaton model that is used in the following chapters. We explain why we want our automata to have multiple heads and nested pebbles, and then give the formal definition with some examples.

Chapters 4 and 5 contain proofs of lemmas which we need for the main theorem. In Chapter 4, we show by induction that for every $\mathrm{FO}(\mathrm{DTC}^k)$ formula there is a corresponding k-head automaton, and describe the changes needed in the proof for the nondeterministic case. In Chapter 5, we consider the other direction, and show that for every deterministic automaton with k heads, there is a corresponding $\mathrm{FO}(\mathrm{DTC}^k)$ formula. The same is done for the nondeterministic automata and $\mathrm{FO}(\mathrm{posTC}^k)$ formulas.

In Chapter 6, we combine the lemmas from the previous two chapters to obtain the main theorem. We also discuss the case of singlehead automata and unary transitive closure, and the nondeterministic version of the main theorem, in which restriction to $\mathrm{FO}(\mathrm{posTC}^k)$ is needed. In Chapter 7, we look at the main theorem and its nondeterministic version in the context of some other related results and open questions.

Chapter 2

Preliminaries

In this chapter, we give definitions of transitive closure logics and word models with some examples. We also explain some of the reasons for adding transitive closure to first-order logic and studying word models. Most of the definitions and examples in this chapter are based on [4].

2.1 Transitive Closure Logic

As logics can be used to describe computations, it is possible to approach complexity theory from the point of view of logic: to characterize queries in a given complexity class by a suitable logic, which has enough expressive power. By itself, first-order logic FO is not quite suitable for this, as some features of computation cannot be expressed in FO. For example, the addition operator we will use in example 2.13 cannot be defined in FO. For this reason, several extensions of first-order logic are studied. Two examples of such extensions are *transitive closure logic* FO(TC) and *deterministic transitive closure logic* FO(DTC), which we obtain by adding to first-order logic the operations of taking *transitive closure* and *deterministic transitive closure* of definable relations. Transitive closure and deterministic transitive closure of a given relation are, in general, inexpressible in first-order logic.

Definition 2.1 (Transitive Closure)**.** Let $k \geq 1$ and X be a $2k$-ary relation on a set B. The *transitive closure* $\mathrm{TC}(X)$ of the relation X is defined by

$$\mathrm{TC}(X) := \{(\bar{a}, \bar{b}) \in B^{2k} : \text{there exists } n > 0 \text{ and } \bar{e}_0, \ldots \bar{e}_n \text{ such that } \bar{a} = \bar{e}_0, \bar{b} = \bar{e}_n, \text{ and for all } i < n,\ (\bar{e}_i, \bar{e}_{i+1}) \in X\},$$

and the *deterministic transitive closure* $\mathrm{DTC}(X)$ by

$$\mathrm{DTC}(X) := \{(\bar{a}, \bar{b}) \in B^{2k} : \text{there exists } n > 0 \text{ and } \bar{e}_0, \ldots \bar{e}_n \text{ such that } \bar{a} = \bar{e}_0, \bar{b} = \bar{e}_n, \text{ and for all } i < n,\ \bar{e}_{i+1} \text{ is the unique } \bar{e} \text{ for which } (\bar{e}_i, \bar{e}) \in X\}.$$

Example 2.2. Let B be a set of some cities in the world and X be a binary relation on a set B such that

$$X = \{(a, b) \in B^2 : \text{airline } A \text{ flies from } a \text{ to } b \text{ without stopover}\}.$$

The transitive closure $\mathrm{TC}(X)$ contains all the pairs (a, b) of the cities in B such that one can fly from a to b (possibly with several stopovers) using only flights of airline A.

Next we define formulas, free variables, and semantics for transitive closure logic FO(TC). For deterministic transitive closure logic FO(DTC) these definitions are analogous.

Definition 2.3 (Transitive Closure Logic)**.** Let τ be a vocabulary. In the following definition, φ and ψ are formulas of vocabulary τ. By FO(TC), we denote the class of formulas of vocabulary τ defined as follows:

(i) All first-order atomic formulas φ are in FO(TC).

(ii) For all formulas φ and ψ in FO(TC), formulas $\neg\varphi$, $\varphi \wedge \psi$, and $\varphi \vee \psi$ are in FO(TC).

(iii) For all variables x and formulas φ in FO(TC), formulas $\exists x\varphi$, and $\forall x\varphi$ are in FO(TC).

(iv) For all tuples $\bar{x}, \bar{y}, \bar{s}, \bar{t}$ of the same length, with tuples of variables $\bar{x}, \bar{y}$ being pairwise distinct and $\bar{s}, \bar{t}$ being tuples of terms, and for all formulas φ in FO(TC), formula $[\mathrm{TC}_{\bar{x},\bar{y}}\ \varphi]\bar{s}\bar{t}$ is in FO(TC).

Let $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_n)$ for some $n \geq 1$. In longer formulas, we will often use the abbreviation $\bar{x} = \bar{y}$ to denote the formula:

$$\bigwedge_{1 \leq i \leq n} x_i = y_i.$$

For a FO(TC) formula φ, the set of free variables of φ is denoted by $\mathrm{free}(\varphi)$. Free variables of FO(TC) formulas are defined in the usual way with the following addition for formulas containing transitive closure:

$$\mathrm{free}([\mathrm{TC}_{\bar{x},\bar{y}}\ \varphi]\bar{s}\bar{t}) = \mathrm{free}(\bar{s}) \cup \mathrm{free}(\bar{t}) \cup (\mathrm{free}(\varphi) \backslash \{\bar{x}, \bar{y}\}).$$

Formulas without free variables are called *sentences*.

Let $\mathcal{B}$ be a τ-model with $\mathrm{Dom}(\mathcal{B}) = B$, and let f be an assignment. The semantics for FO(TC) is defined as the semantics for first-order logic but with the additional definition for transitive closure:

$$\mathcal{B} \models_f [\mathrm{TC}_{\bar{x},\bar{y}}\, \varphi]\bar{s}\bar{t} \quad \text{if and only if} \quad (f(\bar{s}), f(\bar{t})) \in \mathrm{TC}(X_\varphi),$$

where

$$X_\varphi := \{(\bar{a}, \bar{b}) \in B^{2k} : \mathcal{B} \models_{f(\bar{a}/\bar{x}, \bar{b}/\bar{y})} \varphi\}.$$

Next we introduce some notations which will be useful: let τ be a vocabulary and $\mathcal{L}$ a logic. By $\mathcal{L}[\tau]$, we denote the class of formulas of $\mathcal{L}$ of vocabulary τ. For a sentence $\varphi \in \mathcal{L}[\tau]$, the class of finite models of φ is denoted by $\mathrm{Mod}(\varphi)$.

Definition 2.4. Let $\mathcal{L}$ and $\mathcal{L}'$ be logics.

(i) $\mathcal{L} \leq \mathcal{L}'$ if for every τ and every sentence $\varphi \in \mathcal{L}[\tau]$ there exists a sentence $\varphi' \in \mathcal{L}'[\tau]$ such that $\mathrm{Mod}(\varphi) = \mathrm{Mod}(\varphi')$.

(ii) $\mathcal{L} \equiv \mathcal{L}'$ if $\mathcal{L} \leq \mathcal{L}'$ and $\mathcal{L}' \leq \mathcal{L}$.

(iii) $\mathcal{L} < \mathcal{L}'$ if $\mathcal{L} \leq \mathcal{L}'$ and not $\mathcal{L}' \leq \mathcal{L}$.

If $\mathcal{L} \leq \mathcal{L}'$, we say that $\mathcal{L}$ is at most as expressive as $\mathcal{L}'$. Similarly, if $\mathcal{L} \equiv \mathcal{L}'$, we say that logics $\mathcal{L}$ and $\mathcal{L}'$ have the same expressive power.

Example 2.5. For every FO(DTC) formula of the form $\varphi = [\mathrm{DTC}_{\bar{x},\bar{y}}\, \psi(\bar{x}, \bar{y}, \bar{u})]\bar{s}\bar{t}$, there exists an equivalent FO(TC) formula

$$\varphi' = [\mathrm{TC}_{\bar{x},\bar{y}}\, (\psi(\bar{x}, \bar{y}, \bar{u}) \wedge \forall \bar{z}(\neg\psi(\bar{x}, \bar{z}, \bar{u}) \vee \bar{z} = \bar{y})]\bar{s}\bar{t},$$

from which it follows that $\mathrm{FO(DTC)} \leq \mathrm{FO(TC)}$.

Next we define $\mathrm{FO(TC}^k)$ and $\mathrm{FO(DTC}^k)$, which are fragments of transitive closure logic. These fragments and their expressive power will be our main focus in the following chapters.

Definition 2.6 ($\mathrm{FO(TC}^k)$ and $\mathrm{FO(DTC}^k)$)**.** Let $k \geq 1$ and φ be a FO(TC) formula of the form $[\mathrm{TC}_{\bar{x},\bar{y}}\, \psi]\bar{s}\bar{t}$. If the length of tuple $\bar{x}$ is k, formula φ is called the k-ary transitive closure of ψ. We denote by $\mathrm{FO(TC}^k)$ the class FO(TC) formulas for which every subformula of the form φ is a k-ary transitive closure. The class $\mathrm{FO(DTC}^k)$ is defined analogously.

Note that from example 2.5 we immediately obtain that $\mathrm{FO}(\mathrm{DTC}^k) \leq \mathrm{FO}(\mathrm{TC}^k)$, for all $k \geq 1$: if the formula φ in example 2.5 is a k-ary transitive closure then the equivalent formula φ' is also a k-ary transitive closure.

Example 2.7. For all $k \geq 1$ and $\mathrm{FO}(\mathrm{TC}^k)$ formula of the form $[\mathrm{TC}_{\bar{x},\bar{y}}\ \psi]\bar{s}\bar{t}$, there exists an equivalent $\mathrm{FO}(\mathrm{TC}^{k+1})$ formula

$$[\mathrm{TC}_{\bar{x}',\bar{y}'}\ (\psi \wedge x_{k+1} = y_{k+1})]\bar{s}'\bar{t}',$$

where $\bar{u} = (u_1, \ldots, u_k)$ and $\bar{u}' = (u_1, \ldots, u_k, u_{k+1})$, for $u = x, y, s, t$. Note that the same holds analogously for $\mathrm{FO}(\mathrm{DTC}^k)$ formulas, and thus

$$\mathrm{FO}(\mathrm{TC}^k) \leq \mathrm{FO}(\mathrm{TC}^{k+1}) \qquad \text{and} \qquad \mathrm{FO}(\mathrm{DTC}^k) \leq \mathrm{FO}(\mathrm{DTC}^{k+1}).$$

Definition 2.8 ($\mathrm{FO}(\mathrm{posTC})$)**.** An occurrence of transitive closure $[\mathrm{TC}_{\bar{x},\bar{y}}\ \psi]\bar{s}\bar{t}$ is called *positive* if it is in the scope of an even number of negations symbols. The class of $\mathrm{FO}(\mathrm{TC})$ formulas that only have positive occurrences of transitive closure is denoted by $\mathrm{FO}(\mathrm{posTC})$.

2.2 Word Models

In this section, we define a class of finite models called *word models*, but first we introduce some notations that are needed: an alphabet Σ is a finite and non-empty set of symbols. We denote by Σ^* the set of finite strings over Σ. The strings in Σ^* are also called *words*. Note that Σ^* contains the string of length zero, denoted by λ, which we call the *empty word*. The set $\Sigma^+ = \Sigma^* \backslash \{\lambda\}$ is the set of non-empty words. The subsets $L \subseteq \Sigma^*$ are called *languages*.

For some languages, devices called finite automata can be used for checking whether a given word w is in the language. On the other hand, we have logics by which we can describe some properties of structures. Defining the concept of word models makes it possible to formulate a correspondence between words and structures, which is needed for studying the relationship between finite automata and transitive closure logics.

The following definitions and some of the examples are based on [4], although there are some differences which will be explained further after Example 2.10.

Definition 2.9 (Word Model)**.** Let Σ be an alphabet such that $\triangleright, \triangleleft \notin \Sigma$, and define a vocabulary

$$\tau(\Sigma) = \{\leq\} \cup \{P_\alpha : \alpha \in \Sigma \cup \{\triangleright, \triangleleft\}\},$$

where P_α are unary. For $w \in \Sigma^+$, $w = \alpha_1 \ldots \alpha_n$, let structure $\mathcal{B} = (B, \leq, (P_\alpha)_{\alpha \in \Sigma \cup \{\triangleright, \triangleleft\}})$ be such that

(i) $|B| = n + 2$,

(ii) $\leq$ is an ordering of B,

(iii) $P_\triangleright$ is a singleton containing the minimal element of $\leq$,

(iv) $P_\triangleleft$ is a singleton containing the maximal element of $\leq$, and

(v) for all $\alpha \in \Sigma$,

$$P_\alpha = \{b \in B : \text{for some } 1 \leq j \leq n,\ b \text{ is the } j+1\text{-th element of } \leq \text{ and } \alpha_j = \alpha\}.$$

We call such structures *word models* for w, and denote the class of word models for w by K_w.

Example 2.10. Let $\Sigma = \{a, b\}$ and $w = ababab$. The structure

$$(\{0, \ldots, 7\}, \leq, P_\triangleright, P_\triangleleft, P_a, P_b),$$

where $\leq$ is a natural ordering on $\{0, \ldots, 7\}$, $P_\triangleright = \{0\}$, $P_\triangleleft = \{7\}$, $P_a = \{1, 3, 5\}$, and $P_b = \{2, 4, 6\}$, is a word model for w.

Since any two word models for w are isomorphic, it is sufficient to consider the word model of w with the domain $\{0, \ldots, n+1\}$. We call this model *the* word model for w and denote it by $\mathcal{B}_w$.

Note that the definition of a word model differs slightly from the usual (e.g. from the definition in [4]). For a given word w, in addition to the elements corresponding to the positions of the symbols in w, the word model for w contains two extra elements. These elements are always in predicates $P_\triangleright$ and $P_\triangleleft$ as specified in Definition 2.9. When we consider the finite automata in the following chapters, these elements will have corresponding symbols on the input tape that allow the automata to detect the beginning and the end of input words.

There are ways to construct correspondence between automata and structures without having these extra elements in the model. For example, it is possible to do analogously to the case of trees (cf. the child number test and rank of the symbols in [5]), and define the automata to have an additional ability to test whether any of its heads are on the first symbol of the word without the extra symbol marking the start (cf. the case of trees, where the root is the only node with child number 0). In that case, we would have a larger alphabet $\Sigma \cup \{\alpha' : \alpha \in \Sigma\}$, which contains two versions of each letter, and each α' can appear only at the end of a word (cf. the case of trees, where the leaves are labelled by symbols of rank 0). Of course, the instructions of the automaton should also be modified accordingly. This kind of approach is more suited for tree-walking automata which have to

be able to move up and down along the branches of the input tree. In the case of strings, it does not only make the alphabet unnecessarily complicated but also the automata harder to construct. A larger alphabet could be avoided by also adding a test for detecting the end of the input word, but the automaton would have to use the 'beginning' test ('end' test) every time before it moves left (right). This is not that inconvenient because that has to be done in some way anyway, but we still have chosen the more common approach (for two-way automata) to have the two extra symbols on the tape for detecting both ends of the input.

Another way would be to consider the movements of the automaton in the distinctly marked end cells (which do not have the corresponding elements in the word model) in the formulas when we describe computations of the automata by using k-ary transitive closure in Chapter 5. Compared to that, the two extra elements of the model allow us to have less complex formulas – already the formulas for the single computation steps are simpler. It seems likely that proofs similar to the ones in Chapter 5 would also work without the two extra symbols, but it is not clear whether using k-ary transitive closure is enough in that case or do we need higher arity.

Definition 2.11. Let $L \subseteq \Sigma^+$ be a language and $\mathcal{L}$ a logic. The language L *is definable in logic* $\mathcal{L}$ if there is a sentence $\varphi \in \mathcal{L}[\tau(\Sigma)]$ such that

$$\mathrm{Mod}(\varphi) = \bigcup_{w \in L} K_w.$$

Example 2.12. The language Σ^+ is definable in FO. Let ψ_{ord} be the first-order sentence saying that $\leq$ is an ordering, and denote the minimal and the maximal elements of the ordering by min and max. Note that we may introduce these notations because order relations and their minimal and maximal elements are definable in first-order logic. For the following $\mathrm{FO}[\tau(\Sigma)]$-sentence

$$\begin{aligned}\varphi_W :=& \exists x \bigvee_{\alpha \in \Sigma} P_\alpha(x) \wedge \psi_{\mathrm{ord}} \wedge P_{\triangleright}(\min) \wedge P_{\triangleleft}(\max) \wedge \\ & \forall x(x = \min \vee \neg P_{\triangleright}(x)) \wedge \forall x(x = \max \vee \neg P_{\triangleleft}(x)) \wedge \\ & \forall x \bigvee_{\alpha \in \Sigma \cup \{\triangleright, \triangleleft\}} P_\alpha(x) \wedge \bigwedge_{\substack{\alpha, \beta \in \Sigma \cup \{\triangleright, \triangleleft\}, \\ \alpha \neq \beta}} \forall x \neg (P_\alpha(x) \wedge P_\beta(x)),\end{aligned}$$

$\mathrm{Mod}(\varphi_W)$ is the class of all word models.

From this, it follows that the the class of word models is also definable in $\mathrm{FO}(\mathrm{TC}^k)$ and $\mathrm{FO}(\mathrm{DTC}^k)$, and it is reasonable to speak of the definability of languages in these logics. In the following example, we have a language that is definable in $\mathrm{FO}(\mathrm{DTC}^2)$ but not in FO.

Example 2.13. Let $\Sigma = \{a, b\}$ and $L = \{a^n b^n : n \in \mathbb{N}\}$. We show that language L is definable in $\mathrm{FO}(\mathrm{DTC}^2)$.

We want to show that there exists an $\mathrm{FO}(\mathrm{DTC}^2)$ sentence φ_L such that the models of the sentence $\varphi_L \wedge \varphi_W$ are exactly the word models for all the words of L. For this, we again denote the minimal and the maximal elements of the ordering $\leq$ by min and max, and let S be the successor relation. Note that the relation S is definable in first-order logic. It can also be shown that the addition relation $+$ is definable in $\mathrm{FO}(\mathrm{DTC}^2)$ (see e.g. [4]), so we can define the following $\mathrm{FO}(\mathrm{DTC}^2)$ formulas:

$$\chi(s_1, s_2, t_1, t_2) := s_1 = \min \wedge s_2 = \max \wedge t_1 + t_1 = s_2 \wedge S(t_1, t_2)$$

and

$$\psi(x_1, x_2, y_1, y_2) := S(x_1, y_1) \wedge S(y_2, x_2) \wedge P_a(y_1) \wedge P_b(y_2).$$

Now the $\mathrm{FO}(\mathrm{DTC}^2)$-sentence

$$\varphi_L := \exists \bar{s} \exists \bar{t} (\chi(\bar{s}, \bar{t}) \wedge [\mathrm{DTC}_{\bar{x}, \bar{y}}\ \psi] \bar{s} \bar{t})$$

is as wanted.

The language $L = \{a^n b^n : n \in \mathbb{N}\}$ is an example of a language that is not definable in logic $\mathrm{FO}(\mathrm{DTC}^1)$. (Note that since $\mathrm{FO} \leq \mathrm{FO}(\mathrm{DTC}^1)$, L is not definable in FO either.) The reason for this is that, in the class of word models, only *regular* languages are definable in $\mathrm{FO}(\mathrm{DTC}^1)$, and the language L is not regular. Actually, the languages definable in $\mathrm{FO}(\mathrm{DTC}^1)$ over strings are exactly the regular languages, and this is also the case for the logic $\mathrm{FO}(\mathrm{TC}^1)$. (See [1, 8], and Chapter 6 of [4] for more about regular languages.) Although in the case $k = 1$, logic $\mathrm{FO}(\mathrm{DTC}^{k+1})$ has more expressive power than $\mathrm{FO}(\mathrm{DTC}^k)$, it is not known whether $\mathrm{FO}(\mathrm{DTC}^k) < \mathrm{FO}(\mathrm{DTC}^{k+1})$ holds on ordered structures more generally for any k. In other words, we do not know whether there is some arity k_0 such that for all k, we can express every $\mathrm{FO}(\mathrm{DTC}^k)$ formula using only $\mathrm{FO}(\mathrm{DTC}^{k_0})$ formulas [1, 6].

Chapter 3

Two-way Multihead Automata with Nested Pebbles

Finite automata are machines that, given a finite string[1] of symbols as input, can either accept or reject the string. An automaton does this by executing a computation which depends on the input and the instructions of the automaton. During its computation, the automaton goes through a sequence of *states* and its result is based on the last state of the sequence[2].

There exist different types of finite automata. For example, an automaton may only be able to read the input string from left to right, or it may read the symbols of the string with one or more heads which can be moved left and right. The automaton can also be either deterministic or nondeterministic, depending on the instructions. In addition to these examples, there are also automata that are able to mark positions in the string during computation.

The automaton we will consider here and in the following chapters is based on the automaton described in [5]. Before defining the type of automata in detail, we give an informal description of the automaton we are going to work with. In the next section, we also explain reasons why we consider a type of automaton that has multiple heads and nested pebbles.

[1]We only consider finite automata that run with strings as inputs. Generally, the inputs given to automata do not need to be strings. There are also e.g. automata called *tree-walking automata*, which take ordered trees as inputs. For these automata, strings can be thought as a special case of ordered trees called *monadic trees*.

[2]For the automaton that we consider, it is possible that the computation does not halt on some inputs. For our purposes, these computations can be viewed as rejecting, even though we define accepting or rejecting only for halting configurations in Section 3.2.

3.1 On the Properties of the Automaton

Since examining the expressive power of logics $\mathrm{FO}(\mathrm{TC}^k)$ and $\mathrm{FO}(\mathrm{DTC}^k)$ is one of our focuses, we want to have an automaton with the ability to check input strings (words) for properties that can be described with k-ary transitive closure (as the properties of the corresponding word models). For this reason, we will look at a type of automaton which we call *two-way multihead automaton with nested pebbles*. It is a finite automaton that has multiple heads, all of which can be moved in two ways – both left and right – and a finite set of pebbles which can be dropped to mark specific positions in the string.

As the automaton is a two-way one, we may visualize it as moving its heads left and right on a tape where the input word is written. We do not want the automaton to move its heads out of the part of the tape which contains the symbols of the word, so the input is written with two extra symbols marking the start (symbol $\triangleright$) and the end (symbol $\triangleleft$) of the word.

The operation of taking k-ary transitive closure is reflected in the number of the heads of the automaton. As mentioned before, the language $L = \{a^n b^n : n \in \mathbb{N}\}$ is a nonregular language. Two-way *single-head* automata (whether deterministic or nondeterministic) can only accept regular languages (see e.g. Theorem 5 in [1] or [12]), so there exists no such automaton accepting the language L. Since it was shown in example 2.13 that the language L is definable in logic $\mathrm{FO}(\mathrm{DTC}^2)$, we need more than one head to capture deterministic k-ary transitive closure for $k \geq 2$. This is also the case for the nondeterministic k-ary transitive closure because it is at least as expressive as the deterministic version.

On the other hand, when any finite number of heads is allowed, the languages accepted by deterministic two-way multihead automata are exactly the string languages definable in logic $\mathrm{FO}(\mathrm{DTC})$, see e.g. Corollary 3.5 in [9].[3] As we are looking for automata that capture $\mathrm{FO}(\mathrm{DTC}^k)$, an unrestricted number of heads gives us too much computational power.

It can be shown that languages accepted by deterministic and nondeterministic two-way automata with exactly k heads can be described by $\mathrm{FO}(\mathrm{DTC}^k)$ and $\mathrm{FO}(\mathrm{TC}^k)$ formulas, respectively. For that, we do not need the whole classes of $\mathrm{FO}(\mathrm{DTC}^k)$ and $\mathrm{FO}(\mathrm{TC}^k)$ formulas. In the nondeterministic case, such formulas are called *k-regular*, and they capture the power of k-head automata. (For the proof and the definition of k-regular formulas, see [1].) In the deterministic case, the formulas cannot be given in a nice form as in k-regular formulas.

To construct automata for the full class of $\mathrm{FO}(\mathrm{DTC}^k)$ formulas and the class of $\mathrm{FO}(\mathrm{posTC}^k)$ formulas, the two-way k-head automaton model is augmented with pebbles.

[3] Recall that the string languages definable in $\mathrm{FO}(\mathrm{DTC})$ are characterized by the complexity class $\textsc{DSpace}(\log n)$.

This means that the automaton has a finite number of pebbles that can be dropped on the input tape to mark positions. For any position of the tape that is currently scanned by some head, the automaton can test for any pebble whether the pebble is present or not.

The way the pebbles can be picked up has to be restricted: if any pebble can be picked up from the tape at any time, the automaton becomes too powerful again. In the case of deterministic automata, we obtain the languages definable in FO(DTC) and in the nondeterministic case, the languages definable in FO(TC). In order to avoid this, the pebbles of the automaton are nested, which means that only the pebble that was dropped last can be picked up. However, this pebble can be 'retrieved from a distance': in order for the automaton to pick the pebble up, it is not required that the position of the pebble is scanned by any of the heads. In [2], it has been shown that allowing pebbles to be retrieved from a distance does not change the expressive power of the automata.

3.2 Definition and Examples

Definition 3.1. A two-way k-head automaton with nested pebbles (or a 2PA^k) M is a tuple $(Q, \Sigma, X, q_0, A, I)$, where:

(i) Q is a finite set of states,

(ii) Σ is a finite and non-empty alphabet such that $\triangleright, \triangleleft \notin \Sigma$,

(iii) X is a finite set of pebbles,

(iv) $q_0 \in Q$ is the initial state,

(v) $A \subseteq Q$ is the set of accepting states, and

(vi) I is a finite set of instructions.

The instructions in the set I are triples of the form $\langle p, \psi, q\rangle$, $\langle p, \varphi, q\rangle$, or $\langle p, \neg\varphi, q'\rangle$, where $p, q, q' \in Q$ are states, ψ is an operation and φ is a test. Instruction $\langle p, \psi, q\rangle$ means that when the automaton is in state p, it executes the operation ψ, and then changes its state to q. Instructions $\langle p, \varphi, q\rangle$ and $\langle p, \neg\varphi, q'\rangle$ mean that when the automaton is in state p, it does the test φ, and, depending on the result, changes its state to q (affirmative result) or q' (negative result).

The automaton M is deterministic if for any two different instructions $\langle p, \chi_1, q\rangle$ and $\langle p, \chi_2, q'\rangle$, either $\chi_1 = \neg\chi_2$ or $\chi_2 = \neg\chi_1$. In the case that a 2PA^k is deterministic, we use the notation D2PA^k. We sometimes use analogous notation N2PA^k for a nondeterministic 2PA^k to clarify that the automaton is indeed nondeterministic.

There are two different types of operations: head moves and pebble operations. Since the automaton can move any of its heads left and right, for each head i, $1 \leq i \leq k$, there are the head move operations left_i and right_i. The automaton can also drop and retrieve pebbles, so for each head i and pebble $x \in X$, we have the pebble operations $\text{drop}_i(x)$ and $\text{retrieve}(x)$. The pebble operation $\text{retrieve}(x)$ does not refer to any head i because, as mentioned earlier, the pebbles of the automaton are such that they can be retrieved even when no head is scanning the position of the pebble.

The tests are different from the operations in that the tests can appear in instructions also in negated form. If a negated test appears in an instruction, it means that the instruction is applied in the case that the result of the test is negative. The automaton has two types of tests for acquiring information about the existence of certain symbols or pebbles in the current positions of its heads: $\text{symb}_{i,\alpha}$ and $\text{peb}_i(x)$, where $1 \leq i \leq k$ and $\alpha \in \Sigma \cup \{\triangleright, \triangleleft\}$. With $\text{symb}_{i,\alpha}$ and $\neg\text{symb}_{i,\alpha}$ the automaton can test whether the position in which the head i is has the symbol α or not. Similarly with $\text{peb}_i(x)$ and $\neg\text{peb}_i(x)$ the automaton can test the position of the head i for pebble x.

A configuration of M on word $w \in \Sigma^+$ is a triple $[p, \bar{u}, \sigma]$, where $p \in Q$ is a state, $\bar{u} \in \{0, \ldots, |w|+1\}^k$ is a k-tuple indicating the positions of the k heads, and $\sigma = (x_1, v_1) \ldots (x_m, v_m)$ is a stack of pebbles dropped at their positions ($m \geq 0$, $x_j \in X$, $v_j \in \{0, \ldots, |w|+1\}$). The initial configuration is $[q_0, \bar{0}, \varepsilon]$, where q_0 is the initial state, $\bar{0}$ is a k-tuple of 0's indicating that the heads are scanning the position of the start symbol, and ε is the empty stack. Recall that the input is written in the tape with two extra symbols, $\triangleright$ and $\triangleleft$, marking the start and the end of the word, i.e. the positions 0 and $|w|+1$.

The semantics of the automaton is defined with the relation $\vdash_{M,w}$ on configurations for automaton M on input w as follows:

$$[p, \bar{u}, \sigma] \vdash_{M,w} [q, \bar{u}', \sigma'], \text{ where } \sigma = (x_1, v_1) \ldots (x_m, v_m)$$

if there exists an instruction $\langle p, \chi, q \rangle$ such that

if	then
$\chi = \text{left}_i$	$u'[i] = u[i] - 1, u'[h] = u[h]$ for, $h \neq i$, and $\sigma' = \sigma$
$\chi = \text{right}_i$	$u'[i] = u[i] + 1, u'[h] = u[h]$ for, $h \neq i$, and $\sigma' = \sigma$
$\chi = \text{drop}_i(x)$	$\bar{u}' = \bar{u}$, and $\sigma' = \sigma(x, u[i]), x \notin \{x_1, \ldots, x_m\}$
$\chi = \text{retrieve}(x)$	$\bar{u}' = \bar{u}, \sigma'(x_m, v_m) = \sigma$, and $x = x_m, m \geq 1$
$\chi = \text{symb}_{i,\alpha}$	$\bar{u}' = \bar{u}, \sigma' = \sigma$, and $u[i]$ has symbol α
$\chi = \neg\text{symb}_{i,\alpha}$	$\bar{u}' = \bar{u}, \sigma' = \sigma$, and $u[i]$ does not have symbol α
$\chi = \text{peb}_i(x)$	$\bar{u}' = \bar{u}, \sigma' = \sigma$, and $(x, u[i])$ occurs in σ
$\chi = \neg\text{peb}_i(x)$	$\bar{u}' = \bar{u}, \sigma' = \sigma$, and $(x, u[i])$ does not occur in σ

A configuration c is halting if there is no c' such that $c \vdash_{M,w} c'$. A halting configuration c is accepting, if $c = [p, \bar{0}, \varepsilon]$, for some $p \in A$, and otherwise rejecting. Note that this means that when the automaton accepts, it retrieves the pebbles and moves all of its heads back to the position 0. This simplifies proofs in later chapters.

The language $L(M) \subseteq \Sigma^+$ accepted by a $2\mathrm{PA}^k$ M is a set of all nonempty words over Σ on which M has a computation starting from the initial configuration and ending with an accepting configuration, i.e.

$$L(M) = \{w \in \Sigma^+ : ([q_0, \bar{0}, \varepsilon], c) \in \mathrm{TC}(\vdash_{M,w}) \text{ for some accepting configuration } c\}.$$

For every $w \in L(M)$, there must be at least one computation on w that ends with an accepting configuration. If $w \notin L(M)$, each computation on w may or may not halt, but clearly all the halting ones have to end in a rejecting configuration. If M is deterministic, there is only one computation on each word $w \in \Sigma^+$.

Example 3.2. Let $\Sigma = \{a, b\}$ and $L = \{(ab)^n : n \in \mathbb{N}\}$. Now there exists a deterministic single-head automaton M such that $L(M) = L$.

We define M as follows: $M = (Q, \Sigma, X, q_0, A, I)$, where $Q = \{0, \ldots, 9, 9'\}$, $X = \emptyset$, $q_0 = 0$, $A = \{9\}$, and

$$\begin{aligned} I = \{&\langle 0, \mathrm{right}, 1\rangle, \langle 1, \mathrm{symb}_a, 2\rangle, \langle 1, \neg\mathrm{symb}_a, 9'\rangle, \langle 2, \mathrm{right}, 3\rangle, \\ &\langle 3, \mathrm{symb}_b, 4\rangle, \langle 3, \neg\mathrm{symb}_b, 9'\rangle, \langle 4, \mathrm{right}, 5\rangle, \\ &\langle 5, \mathrm{symb}_a, 2\rangle, \langle 5, \neg\mathrm{symb}_a, 6\rangle, \\ &\langle 6, \mathrm{symb}_\triangleleft, 7\rangle, \langle 6, \neg\mathrm{symb}_\triangleleft, 9'\rangle, \langle 7, \mathrm{left}, 8\rangle, \\ &\langle 8, \mathrm{symb}_\triangleright, 9\rangle, \langle 8, \neg\mathrm{symb}_\triangleright, 7\rangle\}, \end{aligned}$$

where the head number in the instructions is omitted as the automaton M has only one head. Note that the finite set of pebbles X is empty because the automaton does not require any pebbles to accept the language L. The language is actually also definable in FO without transitive closure. M halts on every input word $w \in \Sigma^+$, and for each halting configuration $c = [p, u, \sigma]$, state p is either 9 or $9'$. If $p = 9$, then $u = 0$ and $\sigma = \varepsilon$, so the halting configuration c is accepting. If $p = 9'$, the halting configuration c is rejecting.

Example 3.3. Let $\Sigma = \{a, b\}$ and $L = \{a^n b^n : n \in \mathbb{N}\}$. We define a deterministic two-head pebble automaton $M := (Q, \Sigma, X, q_0, A, I)$, where $Q = \{0, \ldots, 14, 14'\}$, $X = \{x\}$,

$q_0 = 0$, $A = \{14\}$, and

$$\begin{aligned} I = \{&\langle 0, \mathrm{right}_1, 1\rangle, \langle 1, \mathrm{right}_1, 2\rangle, \langle 2, \mathrm{symb}_{1,b}, 4\rangle, \langle 2, \neg\mathrm{symb}_{1,b}, 3\rangle, \\ &\langle 3, \mathrm{symb}_{1,\triangleleft}, 14'\rangle, \langle 3, \neg\mathrm{symb}_{1,\triangleleft}, 1\rangle, \\ &\langle 4, \mathrm{drop}_1(x), 5\rangle, \langle 5, \mathrm{right}_2, 6\rangle, \\ &\langle 6, \mathrm{symb}_{2,a}, 7\rangle, \langle 6, \neg\mathrm{symb}_{2,a}, 14'\rangle, \langle 7, \mathrm{right}_1, 8\rangle, \langle 8, \mathrm{right}_2, 9\rangle, \\ &\langle 9, \mathrm{symb}_{1,\triangleleft}, 10\rangle, \langle 9, \neg\mathrm{symb}_{1,\triangleleft}, 11\rangle, \\ &\langle 10, \mathrm{peb}_2(x), 13\rangle, \langle 10, \neg\mathrm{peb}_2(x), 14'\rangle, \\ &\langle 11, \mathrm{symb}_{1,b}, 12\rangle, \langle 11, \neg\mathrm{symb}_{1,b}, 14'\rangle \\ &\langle 12, \mathrm{symb}_{2,a}, 7\rangle, \langle 12, \neg\mathrm{symb}_{2,a}, 14'\rangle, \langle 13, \mathrm{retrieve}(x), 14\rangle\}. \end{aligned}$$

M halts on every input word $w \in \Sigma^+$, and for each halting configuration $c = [p, \bar{u}, \sigma]$, state p is either 14 or $14'$. If $p = 14'$, the halting configuration c is rejecting. The automaton M is such that it uses its first head to find the position where the symbol b appears the first time in the input word, and drops the pebble x in that position. After that, M moves its second head to the position 1, checks that it has symbol a, and starts moving both of its heads right. After each move, M checks whether the first heads reads b and the second head reads a. If any of the heads read a wrong symbol, M rejects. If the first head reaches the end of the word at the same time the second head reaches the position of the pebble x, M accepts.

Note that to be precise, according to our definition, computations of M ending in configurations c, for $p = 14$ and $\sigma = \varepsilon$ are not accepting because $\bar{u} \neq \bar{0}$, i.e. after retrieving the pebble x, M's heads are not in the position 0. Instructions to move the heads to the position 0 can easily be added to the set I, so they are left out only for the sake of simplicity. Assuming that such instructions are added (with the necessary modifications to the sets Q and A), we obtain automaton M', such that $L(M') = L$.

Chapter 4

Constructing Automata for Formulas

In this chapter, we show that for any language definable in $\mathrm{FO}(\mathrm{DTC}^k)$, there is a corresponding $\mathrm{D2PA}^k$ that accepts the language. We also show that the same holds for logic $\mathrm{FO}(\mathrm{posTC}^k)$ and nondeterministic pebble automata, $\mathrm{N2PA}^k$. We first look at the deterministic case. As the deterministic and nondeterministic cases share many similarities, in the nondeterministic case, we focus on describing the modifications that are needed in the proof to handle the nondeterminism. This chapter is based on Section 4 of [5]. Note that the result in [5] is more general as it is for ordered trees, of which word models (or strings) are a special case.

Lemma 4.1. *Let $k \geq 1$ and L be a language definable in $\mathrm{FO}(\mathrm{DTC}^k)$. Then there exists a $\mathrm{D2PA}^k$ that accepts L.*

Proof. As the language L is definable in $\mathrm{FO}(\mathrm{DTC}^k)$, there exists a sentence $\varphi \in \mathrm{FO}(\mathrm{DTC}^k)$ such that the models of φ are exactly the word models for the words in L. The proof is by induction on the structure of the formula: for each $\mathrm{FO}(\mathrm{DTC}^k)$ formula we construct a corresponding $\mathrm{D2PA}^k$. The proof is divided into sections for better readability.

Since not every subformula of φ is necessarily a sentence, we need to have a way to deal with possible free variables in the formulas. We do this by placing pebbles on the input tape. For each free variable of the formula, we add a pebble which marks a position on the tape. The automata we construct will use the positions of these pebbles to check the truth value of the corresponding formula under evaluation indicated by these positions.

In each step, we construct an automaton that will halt on every input. Our automata must always halt because for some formulas φ, the automaton we construct has automata corresponding to the subformulas of φ as subroutines. There are computations that have to halt in an accepting state even when some of these subroutines may reject. In

these cases, the automata corresponding to the subformulas cannot loop because the computation must continue (and eventually halt) after a rejecting subroutine.

Additionally, we construct the automata in a way that every halting configuration (whether accepting or rejecting) will be such that all the heads of the automaton are pointing to the cell in the position 0 on the input tape. This simplifies the constructions as the heads are always in the same positions after a subroutine has halted.

4.1 Atomic Formulas

For atomic formulas, the automata we construct only use one head. We have the atomic formulas: $x = y$, $x \leq y$, and $P_\alpha(x)$, for all $\alpha \in \Sigma \cup \{\triangleright, \triangleleft\}$. We define automata for these formulas as follows:

For formula $x = y$, the automaton moves its head from left to right, checking for each cell whether the pebble x is present or not. When the automaton finds the cell where the pebble x is, it keeps its head in that cell and checks whether the pebble y is also present. If it is, the automaton moves its head to the cell in the position 0 by moving its head left and checking for the symbol $\triangleright$, which indicates that the head is in the correct position. When the head is in the position 0, the automaton halts in an accepting state. If the pebble y is not present in the same cell as the pebble x, the automaton moves its head to the cell in the position 0 and halts in a rejecting state.

For formula $x \leq y$, the automaton moves its head from left to right until it finds the cell where the pebble y is present. When the automaton finds the cell where the pebble y is, it keeps its head in that cell, and first checks whether the pebble x is present or not. Then it checks whether the cell has the symbol $\triangleright$ or not. If neither the pebble x or the symbol $\triangleright$ was found, the automaton keeps moving its head left until it either finds a cell where the pebble x is present or ends up in the cell in the position 0, marked by the symbol $\triangleright$. If the automaton finds the pebble x, it then moves to the cell in the position 0, and halts in accepting state. If the automaton ends up in the cell in the position 0 without finding the pebble x, it halts in a rejecting state.

For formulas $P_\alpha(x)$, $\alpha \in \Sigma \cup \{\triangleright, \triangleleft\}$, the automaton first moves its head right until it finds the cell where the pebble x is present. When the head is in that cell, the automaton checks whether the cell has the symbol α or not. If it does, the automaton moves its head to the cell in the position 0, and accepts. If it does not, the automaton moves its head to the cell in the position 0, and rejects.

4.2 Negation, Conjunction, and Disjunction

In the following constructions of the automata for the negation, conjunction, and disjunction, we assume that we already have the automata for the subformulas ψ_1 and ψ_2.

For the negation $\varphi = \neg\psi_1$ of a formula, the automaton is the same as the automaton for formula ψ_1, but the set A of accepting states is changed to its complement $Q \backslash A$. For this, the automaton for ψ_1 must always halt. As mentioned earlier, for each formula, we build an automaton that halts with the heads being in the cell in the position 0. This ensures that changing the set of accepting states to its complement actually switches rejecting results into accepting ones. Otherwise the heads of the automaton could be in the wrong positions for the halting configuration to be accepting.

For the conjunction $\varphi = \psi_1 \wedge \psi_2$ of formulas, we use the two automata for formulas ψ_1 and ψ_2. First we run the automaton for ψ_1. If this automaton halts in a rejecting state, the automaton for φ also halts and rejects. If the automaton for ψ_1 halts in an accepting state, we run the automaton for ψ_2. If this automaton halts in a rejecting state, then the automaton for φ does the same. If the automaton for ψ_2 halts in an accepting state, then the automaton for φ halts in an accepting state.

For the disjunction $\varphi = \psi_1 \vee \psi_2$ of formulas, the construction of the automaton is similar to the case of conjunction. Again, we use the two automata for formulas ψ_1 and ψ_2, and we first run the automaton for ψ_1. Now the case differs from the earlier case of conjunction: if this automaton halts in an accepting state, then so does the automaton for φ. If the automaton halts in a rejecting state, the automaton for φ does not halt yet. Instead, we now run the automaton for ψ_2. If this automaton halts in a rejecting state, then the automaton for φ does the same. If the automaton for ψ_2 halts in an accepting state, then the automaton for φ halts in an accepting state. Note that this construction works because the automaton for ψ_1 always halts, so in the case that the automaton for ψ_1 rejects, it is possible to next run the automaton for ψ_2.

In the cases of conjunction and disjunction, the free variables in ψ_1 and ψ_2 are not necessarily the same, and the automaton for φ moves its heads on the input tape, where there are pebbles present for all of the free variables that appear in ψ_1 or ψ_2. This does not pose any difficulties for us, as the automata we use for subformulas ψ_1 and ψ_2 only look for the pebbles that represent the free variables in each subformula, and the pebbles that are placed by each automaton itself. How the pebbles placed by the automaton affect this, will be clarified in the next section after we have described the automata for universal and existential quantifiers. Handling these quantifiers requires that the automata can place pebbles on the input tape.

4.3 Universal and Existential Quantifiers

In the constructions of automata for universal and existential quantifiers, we need to take into account the fact that formulas containing quantifiers have variables that are bound. The automata we construct will handle these bounded variables by placing a pebble on the input tape for each bounded variable.

As before in the cases of the negation, conjunction, and disjunction, we assume that we already have the automata for the subformulas ψ_1.

For formula $\varphi = \forall x \psi_1$, we construct the automaton for φ as follows. Starting with $v = 0$, the automaton first tests whether the cell in the position v has the symbol $\triangleleft$ or not. If not, the automaton drops the pebble x in the cell in the position v, moves its head back to the position 0, and runs the automaton for ψ_1. The pebble x that was dropped now marks a position that the automaton for ψ_1 uses for variable x. If the automaton for ψ_1 rejects with the placement of pebble x in the position v, then the automaton for φ halts and rejects (first retrieving the pebble x). If the automaton for ψ_1 accepts, the automaton for φ finds the position v of the pebble x again. Then it moves its head to the next cell $v + 1$, and tests it for the symbol $\triangleleft$. If there is no symbol $\triangleleft$, the automaton retrieves the pebble x, and drops it in the cell $v+1$, and runs the automaton for ψ_1 again for this placement of the pebble. Depending on the result, the automaton for φ then proceeds as in the case of the position v. The automaton does this until it either halts in a rejecting state or reaches the cell with the symbol $\triangleleft$ while trying to drop the pebble x. If the cell with the symbol $\triangleleft$ is reached, the automaton drops the pebble x in that cell, and we run the automaton for ψ_1 for the last time. If it rejects, the automaton for φ halts and rejects, as before. If it accepts, the automaton now retrieves pebble x, and halts in an accepting state.

For formula $\varphi = \exists x \psi_1$, we have a similar construction as in the previous case. The difference is that if the automaton for ψ_1 rejects with the placement of pebble x in the position v, then the automaton for φ does not reject – instead, it next runs the automaton for ψ_1 for the placement of pebble x in the position $v + 1$. If there is a placement of the pebble x such that the automaton for ψ_1 accepts, then the automaton for φ halts in an accepting state with the first one of such placements (again, first retrieving the pebble x). If the cell with the symbol $\triangleleft$ is reached, and the automaton for ψ_1 rejects again, then no such placement exists, and the automaton retrieves pebble x, and halts in a rejecting state. Note that as in the case of disjunction, in this construction we use the fact that the automaton for ψ_1 always halts.

Now that we have seen how the automata handle bound variables, we return to the question posed in the previous section. How do the pebbles placed by the automaton affect the cases of conjunction and disjunction? For example, when $\psi_1 = \forall x P_\alpha(x)$ and $\psi_2 = x \leq y$, in the case of conjunction we have formula $\varphi = \forall x P_\alpha(x) \wedge x \leq y$. Now the

automaton for ψ_1 cannot place the pebble x because the pebbles x and y corresponding to the free variables of φ are already placed on the input tape before the computation for φ begins. But this can be fixed by changing the bound appearances of variable x to a fresh variable that does not appear in the formula.

4.4 Transitive Closure

In the construction of automata for a transitive closure formula $\varphi = [\mathrm{DTC}_{\bar{x},\bar{y}}\ \psi_1]\bar{s}\bar{t}$, there are more details we need to consider than in the previous cases. As in the case of the quantifiers, formulas containing transitive closure have variables that are bound. These variables are handled as before, by placing pebbles on the input tape. In addition to this, we have to pay attention to the formula ψ_1 and its free variables.

We assume that ψ_1 has $2k$ free variables $\bar{x}$ and $\bar{y}$. The remaining free variables of ψ_1 are fixed by pebbles before the automaton for φ starts its computation, and since these pebbles are not moved during the computation, we may disregard the corresponding free variables.

For formula $\varphi = [\mathrm{DTC}_{\bar{x},\bar{y}}\ \psi_1]\bar{s}\bar{t}$, we construct an automaton that checks whether it is possible to find a sequence $\bar{v}_0, \ldots, \bar{v}_n$ of k-tuples of positions in the input tape such that $\bar{v}_0 = \bar{s}$, $\bar{v}_n = \bar{t}$, and each pair $(\bar{v}_i, \bar{v}_{i+1})$ of consecutive k-tuples of the sequence satisfies the formula ψ_1. Since we consider the deterministic transitive closure, the sequence should additionally be such that for $i < n$, the pair $(\bar{v}_i, \bar{v})$ satisfies ψ_1 only when $\bar{v} = \bar{v}_{i+1}$.

We define a directed graph G that helps us with the construction of the automaton. Let $\bar{v}$ be a k-tuple of some positions $v_1, \ldots, v_k$ of an input word w (including positions 0 and $|w| + 1$). We place all of such k-tuples in lexicographic order,[1] and let the graph G have a vertex for each k-tuple $\bar{v}$. We add an edge from vertex $\bar{v}$ to vertex $\bar{v}'$ if $\psi_1(\bar{v}, \bar{v}')$ is satisfied in the word model $\mathcal{B}_w$ for w. Note that the automaton will not have direct access to this graph (or rather to the graph G', which we define next). The graph can be thought of as a virtual computation space that the automaton reconstructs during its computation by using the automaton for ψ_1.

Now the existence of a wanted sequence of k-tuples can be thought of as having a path in G connecting the two vertices corresponding to the two k-tuples $\bar{s}$ and $\bar{t}$. We can try to find a path by starting from one vertex and trying possible paths in lexicographic order of the vertices. This cannot be done by starting from the vertex $\bar{s}$ and trying to find a path to the vertex $\bar{t}$ because we may end up looping. If some vertex in graph G has more than one incoming edge, we might go through this vertex more than once, and after that, we

[1] Let B be a finite set of natural numbers. The lexicographic ordering of k-tuples $(v_1, \ldots, v_k)$, where $v_j \in B$ for $1 \leq j \leq k$, is an ordering such that $(v_1, \ldots, v_k) \leq (v'_1, \ldots, v'_k)$ if $v_j = v'_j$ for all $1 \leq j \leq k$ or $v_{j'} < v'_{j'}$, where $1 \leq j' \leq k$ is the first index such that $v_{j'} \neq v'_{j'}$.

will start circling this path without finding a correct one. Instead, we go backwards from the vertex $\bar{t}$, starting with trying to find a vertex $\bar{v}$ such that $\psi_1(\bar{v}, \bar{t})$ holds. We do this by trying to find the path in a new graph G', which we obtain by changing the direction of every edge in G.

We still have the exact same problem with possible looping, but now that we are trying to find a path in the graph G', there is a simple way to fix it. We cannot start looping in this way if there is at most one incoming edge to each vertex (i.e. each vertex of the original graph G has an outdegree of at most 1). For this, we just have to assume that the formula $\psi_1(\bar{x}, \bar{y})$ is *functional*, meaning that for every word model $\mathcal{B}$ and k-tuple $\bar{v}$ of elements of $B = \mathrm{Dom}(\mathcal{B})$, there is at most one $\bar{v}' \in B^k$ such that $\psi_1(\bar{v}, \bar{v}')$ is satisfied in $\mathcal{B}$. We may assume this because for every $\mathrm{FO}(\mathrm{DTC}^k)$ formula $\varphi = [\mathrm{DTC}_{\bar{x},\bar{y}}\, \psi_1(\bar{x}, \bar{y})]\bar{s}\bar{t}$, we have an equivalent formula

$$\varphi' = [\mathrm{DTC}_{\bar{x},\bar{y}}\, (\psi_1(\bar{x}, \bar{y}) \wedge \forall \bar{z}(\neg\psi_1(\bar{x}, \bar{z}) \vee \bar{z} = \bar{y})]\bar{s}\bar{t},$$

where the deterministic transitive closure is now taken respective to a functional formula. As we are looking at languages definable by $\mathrm{FO}(\mathrm{DTC}^k)$ sentences, the exact syntax of the formula φ does not matter in this sense.

Note that if there is an incoming edge to the starting vertex $\bar{t}$, we can still get stuck. If such an incoming edge exists, we can fix this by 'throwing the edge away': when the automaton is trying to find the path in graph G', for any vertices $\bar{v}, \bar{v}'$, after checking if $\psi_1(\bar{v}, \bar{v}')$ holds, the automaton additionally checks whether $\bar{v} \neq \bar{s}$ and $\bar{v} = \bar{t}$ hold[2], and if they do, the automaton ignores the edge from $\bar{v}'$ to $\bar{v}$. This can be done because on the input tape, the cell positions corresponding to the vertices $\bar{s}$ and $\bar{t}$ are marked by pebbles, so the automaton can recognize them. Knowing that such checking can be done, later when describing the automaton, we may assume that there are no incoming edges to the vertex $\bar{t}$.

As mentioned before, the automaton for φ places pebbles on the input tape and uses these pebbles to evaluate the free variables of the formula ψ_1. Since we want to find a path in graph G' by checking the existence of edges in lexicographic order of the vertices, the automaton has to be able to place pebbles to positions on the input tape in a way that corresponds to the lexicographic order of vertices.

For that reason, before fully describing the automaton for φ, we show how the automaton can go through the placements of pebbles $x_1, \ldots, x_k$ to the cells in positions $v_1, \ldots, v_k$ in lexicographic order. As the automaton starts with all the heads in the position 0, it can place the pebbles $x_1, \ldots, x_k$ to the positions indicated by the first k-tuple $(0, \ldots, 0)$ of the lexicographic ordering by just dropping them one by one, starting with x_1. Let

[2]Checking both that $\bar{v} \neq \bar{s}$ and $\bar{v} = \bar{t}$ is done to handle the special case that $\bar{s} = \bar{t}$. In the case that $\bar{s} \neq \bar{t}$, it would be enough to check only that $\bar{v} = \bar{t}$.

$\bar{v}$ be the k-tuple indicating the positions of the pebbles $x_1, \ldots, x_k$. Next we show how the automaton moves the pebbles to the positions $v'_1, \ldots, v'_k$, where $\bar{v}' = (v'_1, \ldots, v'_k)$ is the successor of $\bar{v}$ in lexicographic order. The automaton first finds the position v_k of the pebble x_k. When the position v_k is found, the automaton checks if the cell v_k has the symbol $\triangleleft$. If not, then the automaton moves its head to the next cell in the position $v_k + 1$, retrieves x_k and drops it in the cell $v_k + 1$. Thus $\bar{v}' = (v_1, \ldots, v_{k-1}, v_k + 1)$ and we are done. If the cell v_k has the symbol $\triangleleft$, then the automaton moves to the cell in the position 0, retrieves x_k, and drops it in the cell 0. Then the automaton finds the position v_{k-1} of the pebble x_{k-1}, and continues in the same way as for the pebble x_k, starting with checking if the cell v_{k-1} has the symbol $\triangleleft$. If the automaton ends up going through all of the pebbles $x_k, \ldots, x_1$, and it cannot move the pebble x_1 to the position $v_1 + 1$, it finds this out by checking that the position v_1 has the symbol $\triangleleft$. This can only happen when the tuple $(v_1, \ldots, v_k)$ is the last one in lexicographic order. In this case the automaton has already gone through all the placements of the pebbles.

Now that it has been shown how the automaton can place the pebbles $x_1, \ldots, x_k$ to the positions $v_1, \ldots, v_k$ in the lexicographic order of the vertices of G', we show how the automaton can find a wanted path in the graph G' (if it exists). The central idea is that when we are in some vertex $\bar{v}'$ on a possible path from $\bar{t}$ to $\bar{s}$, we try to find a next vertex $\bar{v}$ such that there is an edge from $\bar{v}'$ to $\bar{v}$. We start from vertex $\bar{v}' = \bar{t}$, marked by pebbles $\bar{y}$, and use pebbles $\bar{x}'$ and $\bar{y}'$ to find a vertex $\bar{v}$. We do this by first placing the pebbles $\bar{y}'$ to the positions indicated by the vertex $\bar{v}'$, and then placing the pebbles $\bar{x}'$. We place the pebbles $\bar{x}'$ to the positions corresponding to the vertices of G', and go through vertices in lexicographic order until we find a vertex $\bar{v}$ with an incoming edge from $\bar{v}'$. From the vertex $\bar{v}$, we then again try to find a next vertex with an incoming edge from $\bar{v}$ using pebbles $\bar{x}'$ and $\bar{y}'$. We do this until we have found the vertex $\bar{s}$ marked by the pebbles $\bar{x}$ or there are no outgoing vertices from the vertex $\bar{v}$ marked by the pebbles $\bar{y}'$. If there are no outgoing vertices from the vertex $\bar{v}$, we need to go back to the unique vertex $\bar{v}'$ with an outgoing edge to the vertex $\bar{v}$ and try the next vertex (i.e. the successor of $\bar{v}$) in lexicographic order. If we end up going back to the vertex $\bar{t}$ marked by the pebbles $\bar{y}$, and testing all the vertices for possible paths with no luck, we know that a wanted path cannot be found.

There is one important thing to note: when we end up in a vertex $\bar{v}$ with no outgoing edges, in order to try the next vertex of the lexicographic ordering, we need to remember which vertex was the one that we just checked. This can be done because the vertex $\bar{v}$ is marked by the pebbles $\bar{y}'$, but we also need pebbles to find back to the vertex $\bar{v}'$ from which we can look for a 'new' vertex $\bar{v}$. If we use the pebbles $\bar{x}'$ for finding $\bar{v}'$, we do not have any pebbles left for trying to find the 'new' vertex $\bar{v}$, as we cannot move the pebbles $\bar{y}'$ before lifting the pebbles $\bar{x}'$. To keep the pebbles nested, we have to use additional pebbles $\bar{z}'$. As it is necessary to pay attention to the order in which we place and lift the

pebbles, in addition to the main idea of the automaton, we now also give a more detailed description.

The automaton uses $3k$ pebbles $\bar{x}' = x'_1, \ldots, x'_k$, $\bar{y}' = y'_1, \ldots, y'_k$, and $\bar{z}' = z'_1, \ldots, z'_k$. Note that before the actual computation of the automaton starts, the pebbles $\bar{x} = x_1, \ldots, x_k$ and $\bar{y} = y_1, \ldots, y_k$ are already placed on the input tape and the automaton does not move these pebbles. First, the automaton finds the positions $v'_1, \ldots, v'_k$ of the pebbles $\bar{y}$, and places the pebbles $\bar{y}'$ in the same positions. Then, starting with the first k-tuple $\bar{v} = (0, \ldots, 0)$ of the positions $v_1, \ldots, v_k$ of lexicographic ordering, the automaton places k pebbles $\bar{x}'$ to the k cells in the positions $\bar{v}$. Then we run the automaton corresponding to the formula $\psi_1(\bar{x}', \bar{y}')$.

If the automaton for $\psi_1(\bar{x}', \bar{y}')$ accepts, the automaton for φ checks whether $\bar{x}' = \bar{x}$ (i.e. whether the position v_j of the pebble x'_j is the same as the position of the pebble x_j, for $1 \leq j \leq k$). If this holds, the automaton halts and accepts (first retrieving the pebbles and moving its heads to the position 0). If not, the automaton moves each head i (of its k heads) to the position v_i of the pebble x'_i. Then the automaton retrieves the pebbles $\bar{x}'$ and $\bar{y}'$ (from a distance, without moving its heads), and then places each pebble y'_i to the position v_i of the head i. Now these positions mark a 'new' vertex $\bar{v}'$, from which we try find the next vertex $\bar{v}$. So the automaton starts again with the first tuple $\bar{v} = (0, \ldots, 0)$ of the positions of lexicographic ordering, and places pebbles $\bar{x}'$ to the k cells in the positions $\bar{v}$. Then we run the automaton corresponding to the formula $\psi_1(\bar{x}', \bar{y}')$ again.

If the automaton for $\psi_1(\bar{x}', \bar{y}')$ rejects, we move the pebbles $\bar{x}'$ to the next k-tuple of positions in lexicographic order, and run the automaton for $\psi_1(\bar{x}', \bar{y}')$ again for this new placement of pebbles. If the automaton goes through all k-tuples of positions for the pebbles $\bar{x}'$ without the automaton for $\psi_1(\bar{x}', \bar{y}')$ accepting, we have to move the pebbles $\bar{y}'$ to the next k-tuple of positions in lexicographic order. To do this, we need to use the pebbles $\bar{z}'$. The automaton first retrieves pebbles $\bar{x}'$. (The pebbles $\bar{y}'$ are still left on the input tape.) Then the automaton places the pebbles $\bar{z}'$ in positions in lexicographic order of the vertices of G' to find the only vertex with an outgoing edge to the vertex marked by the pebbles $\bar{y}'$. This is done by using the automaton for $\psi_1(\bar{y}', \bar{z}')$. When the placement of the pebbles corresponding to this vertex is found, the automaton finds the positions of the pebbles $\bar{y}'$ again. The automaton continues from there in lexicographic order, placing the pebbles $\bar{x}'$ and using them to find the next vertex by using the automaton for $\psi_1(\bar{x}', \bar{z}')$. When the positions corresponding to the correct vertex are found, the automaton checks whether $\bar{x}' = \bar{x}$, and halts and accepts if this holds (first retrieving the pebbles and moving its heads to the position 0). If it does not, the automaton moves its heads to the positions of the pebbles $\bar{x}'$, retrieves pebbles $\bar{x}'$, $\bar{z}'$, and $\bar{y}'$, and places pebbles $\bar{y}'$ in the positions where the heads are. Then the automaton continues with placing the pebbles $\bar{x}'$ to the first tuple $\bar{v}$ in lexicographic order.

If the automaton does not accept, it ends up placing the pebbles $\bar{y}'$ back to the

vertex marked by pebbles $\bar{y}$ and going through all placements of pebbles $\bar{x}'$ without the automaton for $\psi_1(\bar{x}', \bar{y}')$ accepting. When this happens, the automaton retrieves pebbles $\bar{x}'$ and $\bar{y}'$, moves all of its heads to the position 0, and halts and rejects.

From the previous constructions we see that for each formula $\varphi \in \mathrm{FO}(\mathrm{DTC}^k)$, the corresponding automaton needs only a finite number of pebbles. The number of pebbles depends on the number of nested quantifiers and transitive closures of the formula. We use one pebble for each quantifier and $3k$ for transitive closure, but as the pebbles can be reused, it is enough to have a number of pebbles that is the maximum depth of nested operators appearing in the formula.

4.5 Nondeterministic Automata and FO(posTC^k)

It is not known whether the class of languages accepted by nondeterministic two-way k-head pebble automata is closed under complement, so we cannot handle negations of formulas in the same way we did in the deterministic case. We will have to restrict to languages definable in $\mathrm{FO}(\mathrm{posTC}^k)$. For every formula in $\mathrm{FO}(\mathrm{posTC}^k)$, there is an equivalent formula where negation appears only in front of atomic formulas. Since we can construct automata for negations of atomic formulas, this restriction allows us to prove the following lemma for $\mathrm{FO}(\mathrm{posTC}^k)$ and the nondeterministic automata.

Lemma 4.2. *Let $k \geq 1$ and L a language definable in* $\mathrm{FO}(\mathrm{posTC}^k)$. *Then there exists an* $\mathrm{N2PA}^k$ *that accepts L.*

Proof. For atomic formulas, conjunction, and universal quantification, the constructions of the automata are much like in the deterministic case before. Note that in the cases of conjunction and universal quantification the automata might not halt if the result of the automaton is not accepting. This is not a problem because we do not require the nondeterministic automata to always halt. For every word w in the language L, the automaton only needs to have at least one accepting computation on w.

As the automata for atomic formulas always halt, we get the automata for negations of atomic formulas from the construction in the deterministic case. The automata for disjunction, existential quantification, and (nondeterministic) transitive closure will use nondeterminism, so we will describe the constructions for these automata.

For the disjunction $\varphi = \psi_1 \vee \psi_2$ of formulas, the automaton nondeterministically chooses between two instructions, resulting in running either the automaton for ψ_1 or the automaton for ψ_2. The result of the automaton for φ is the same as the result of the chosen automaton. Now the automaton for φ has a computation on a given word ending in an accepting configuration if and only if the automaton for ψ_1 or the automaton for ψ_2 has one.

For formula $\varphi = \exists x \psi_1$, the automaton first nondeterministically chooses a placement for pebble x. This can be done by having the automaton move its head from left to right and check for each cell whether the cell has the symbol $\triangleleft$. If not, then it either chooses to place the pebble on that cell or to move its head one cell to the right without placing the pebble. If the automaton places the pebble, it moves its head back to the position 0. If the automaton ends up in the cell marked by the symbol $\triangleleft$, it is in the last cell and it has to place the pebble on that cell. After placing the pebble and moving to the position 0, we run the automaton for ψ_1 with the placement chosen for the pebble x. The result of the automaton for φ is the same as the result of the automaton for ψ_1. The automaton for φ has a computation on a given word ending in an accepting configuration if and only if there is some placement of pebble x for which the automaton for ψ_1 has one.

For formula $\varphi = [\mathrm{TC}_{\bar{x},\bar{y}}\, \psi_1]\bar{s}\bar{t}$, we cannot assume that the formula ψ_1 is functional as we did in the deterministic case. Fortunately, this assumption is not needed, because allowing the automaton to be nondeterministic simplifies its construction. By using nondeterminism, the checking for existence of a path can be done in a straightforward way from $\bar{s}$ to $\bar{t}$ in the graph G. The automaton places pebbles $\bar{x}'$ to the positions indicating the current vertex, places the pebbles $\bar{y}'$ to a candidate vertex, and then checks for existence of an edge by using the automaton for $\psi_1(\bar{x}', \bar{y}')$.

Now the placement of the pebbles $\bar{y}'$ is not done in lexicographic order but chosen nondeterministically pebble by pebble as described in the case of existential quantification. Note that the possibility of looping is not a problem because the automaton does not need to halt on every computation.

When the automaton for $\psi_1(\bar{x}', \bar{y}')$ accepts, indicating that an edge was found, the automaton checks whether $\bar{y}' = \bar{y}$. If it does, the automaton accepts. If it does not, the automaton moves its heads to the positions of pebbles $\bar{y}'$, lifts pebbles $\bar{y}'$, and then retrieves $\bar{x}'$ from a distance, without moving its heads. Then the automaton places the pebbles $\bar{x}'$ in the position of the heads, and continues checking for some candidate vertex by again nondeterministically choosing a placement for pebbles $\bar{y}'$. If a path from $\bar{s}$ to $\bar{t}$ in the graph G exists, the automaton for φ finds it on some of its computations.

Chapter 5

Defining Formulas for Automata

In this chapter, we show that for every language accepted by some D2PA^k, there is a FO(DTC^k) formula that defines the language. As in the previous chapter, we also show that the same holds for nondeterministic pebble automata, N2PA^k, and logic FO(posTC^k). We do this by constructing formulas that describe the accepting computations of the automaton: we define formulas for single computation steps of the automaton, and by using k-ary transitive closure, we construct formulas that describe computations consisting of sequences of consecutive steps.

This chapter is mostly based on Section 5 of [5]; some parts are also based on Section 3 of [1] by Bargury and Makowsky. The technique used is similar to Kleene's algorithm [11] that transforms classical finite automata into regular expressions. (For regular expressions and languages, see also [8] and Chapter 6 of [4].) In [1], it was shown that Kleene's algorithm can be generalized to multi-head automata in multidimensional grids, and transitive closure can be used to express sequences of consecutive positions of the heads. As the automata we use here additionally contain pebbles, we follow the proof presented in [5], iterating the construction for each pebble of the automaton.

The deterministic and nondeterministic cases are again similar, so we focus on the deterministic case, and then describe the necessary modifications to the proof in the nondeterministic case. In Section 5.1, we define a matrix which expresses computational behaviour of a given automaton. We call it *the computation closure matrix*, and in Lemma 5.3, we present some observations concerning it. These observations will be useful in Section 5.2, where we show how to define formulas that describe computations of the automaton. The observations are used for proofs in both deterministic and nondeterministic cases.

5.1 Computation Closure Matrix

Let M be a 2PA^k and Q the finite set of its states. We define Φ as a $Q \times Q$ matrix of predicates $\varphi_{p,q}(\bar{x}, \bar{y})$, where $p, q \in Q$, and $\bar{x}, \bar{y}$ are k-tuples of distinct variables occurring free in all predicates in Φ. These predicates can be thought to correspond to the single computation steps[1] of the automaton M. Later in this chapter, the predicates will be defined with suitable formulas, so we assume here that each predicate is definable by a formula. In this chapter, we denote $\mathrm{Dom}(\mathcal{B}) = B$ for any word model $\mathcal{B}$.

Definition 5.1 (Computation closure)**.** Let Φ be as above. The *computation closure* of Φ with respect to $\bar{x}, \bar{y}$ is the matrix $\Phi^{\#}$ consisting of predicates $\varphi^{\#}_{p,q}(\bar{x}, \bar{y})$ such that for any word model $\mathcal{B}$, $\mathcal{B} \models \varphi^{\#}_{p,q}(\bar{u}, \bar{u}')$ if and only if there exists a sequence $\bar{u}_0, \ldots \bar{u}_n$ of k-tuples of elements in B and a sequence $p_0, \ldots, p_n$ of states, such that $n \geq 1$, $\bar{u} = \bar{u}_0$, $\bar{u}' = \bar{u}_n$, $p = p_0$, $q = p_n$, and $\mathcal{B} \models \varphi_{p_i,p_{i+1}}(\bar{u}_i, \bar{u}_{u+1})$ for $0 \leq i < n$.

In other words, the matrix Φ contains predicates that correspond to the *single* computation steps of the automaton M, and the matrix $\Phi^{\#}$ contains predicates that correspond to the *sequences* of these steps. Note that in the definition above, the predicates $\varphi^{\#}_{p,q}(\bar{x}, \bar{y})$ are also assumed to be definable by formulas. Let w be some word, and $\mathcal{B}_w$ the word model for w. Then $\mathcal{B}_w \models \varphi^{\#}_{p,q}(\bar{u}, \bar{u}')$ means that when the automaton M is run on input w, for some n, there is a Φ-path of n consecutive steps of M leading from state p to state q, where in state p, the k heads of M are in positions $\bar{u}$, and in state q, they are in positions $\bar{u}'$. Since we require that $n \geq 1$, the path is always nonempty.

Suppose that in addition to $\bar{x}, \bar{y}$, all the free variables of formulas for predicates $\varphi_{p,q}(\bar{x}, \bar{y})$ are among $z_1, \ldots, z_m$. The formulas for predicates $\varphi^{\#}_{p,q}(\bar{x}, \bar{y})$ will be defined with the help of predicates $\varphi_{p,q}(\bar{x}, \bar{y})$, so the free variables of each $\varphi^{\#}_{p,q}(\bar{x}, \bar{y})$ will be among $\bar{x}, \bar{y}, z_1, \ldots, z_m$. If $\mathcal{B} \models \varphi^{\#}_{p,q}(\bar{u}, \bar{u}', v_1, \ldots, v_m)$, the variables $z_1, \ldots, z_m$ must have fixed values $v_1, \ldots, v_m$.

Definition 5.2. If for any states $p, q, q' \in Q$, word model $\mathcal{B}$, and $\bar{u}, \bar{u}', \bar{u}'' \in B^k$,

(i) $\mathcal{B} \models \varphi_{p,q}(\bar{u}, \bar{u}')$ and $\mathcal{B} \models \varphi_{p,q}(\bar{u}, \bar{u}'')$ implies that $\bar{u}' = \bar{u}''$, then the predicate $\varphi_{p,q}(\bar{x}, \bar{y})$ is *functional*,

(ii) $\mathcal{B} \models \varphi_{p,q}(\bar{u}, \bar{u}')$ and $\mathcal{B} \models \varphi_{p,q'}(\bar{u}, \bar{u}'')$ implies that $q = q'$, then the predicate $\varphi_{p,q}(\bar{x}, \bar{y})$ is *exclusive*.

Matrix Φ is called *deterministic* if its predicates are both functional and exclusive.

[1] Not all of these predicates correspond to any computation step of automaton M, but such predicates will be defined with formulas that are always false.

We say that a state q is *final* if $\mathcal{B} \not\models \varphi_{q,r}(\bar{u}, \bar{u}')$ for any state $r \in Q$, word model $\mathcal{B}$, and $\bar{u}, \bar{u}' \in B^k$. If the requirements (i) and (ii) of definition 5.2 hold for predicates $\varphi_{p,q}(\bar{x}, \bar{y})$ and $\varphi_{p,q'}(\bar{x}, \bar{y})$, only when q and q' are final states, then Φ is called *semi-deterministic*.

In the following Lemma, when we say that the matrix Φ is in some logic $\mathcal{L}$, we mean that all the predicates of the matrix are formulas in $\mathcal{L}$.

Lemma 5.3. *Let Φ be a matrix of predicates and $\Phi^{\#}$ its computation closure.*

(i) If Φ is deterministic, then $\Phi^{\#}$ is semi-deterministic.

(ii) If Φ is in $\mathrm{FO}(\mathrm{TC}^k)$, then $\Phi^{\#}$ can also be defined in in $\mathrm{FO}(\mathrm{TC}^k)$.

(iii) If Φ is in $\mathrm{FO}(\mathrm{DTC}^k)$ and deterministic, then $\Phi^{\#}$ can be defined in $\mathrm{FO}(\mathrm{DTC}^k)$.

Proof. *(i)* Let $q, q' \in Q$ be final states, and $\mathcal{B}$ a word model such that $\mathcal{B} \models \varphi^{\#}_{p,q}(\bar{u}, \bar{u}')$ and $\mathcal{B} \models \varphi^{\#}_{p,q'}(\bar{u}, \bar{u}'')$ for $p \in Q$ and $\bar{u}, \bar{u}', \bar{u}'' \in B^k$. Assume that the lengths of Φ-paths for $\varphi^{\#}_{p,q}(\bar{u}, \bar{u}')$ and $\varphi^{\#}_{p,q'}(\bar{u}, \bar{u}'')$ are n and n', respectively.

Let $n = 1$. Then $\mathcal{B} \models \varphi_{p,p_1}(\bar{u}, \bar{u}_1)$ for $p_1 = q$ and $\bar{u}_1 = \bar{u}'$, and $\mathcal{B} \models \varphi_{p,p'_1}(\bar{u}, \bar{u}'_1)$ for some $p'_1 \in Q$ and $\bar{u}'_1 \in B^k$. From the determinism of Φ, it follows that the predicates $\varphi_{p,p_1}(\bar{x}, \bar{y})$ and $\varphi_{p,p'_1}(\bar{x}, \bar{y})$ are functional and exclusive, so $p'_1 = p_1$ and $\bar{u}'_1 = \bar{u}_1$. Since $p'_1 = q$ is final, $\mathcal{B} \not\models \varphi^{\#}_{p'_1,q'}(\bar{u}'_1, \bar{u}'')$, which means that the path cannot be extended from p'_1 and $\bar{u}'_1$. Thus $n' = n = 1$, implying that $p'_1 = q'$, and $\bar{u}'_1 = \bar{u}''$. Now $q = q'$ and $\bar{u}' = \bar{u}''$, so predicates $\varphi^{\#}_{p,q}(\bar{x}, \bar{y})$ and $\varphi^{\#}_{p,q'}(\bar{x}, \bar{y})$ are functional and exclusive.

Let $n = l + 1$ and $n' = l' + 1$ for some $l, l' \geq 1$. From $\mathcal{B} \models \varphi^{\#}_{p,q}(\bar{u}, \bar{u}')$, it now follows that there exists $p_1 \in Q$ and $\bar{u}_1 \in B^k$ such that $\mathcal{B} \models \varphi_{p,p_1}(\bar{u}, \bar{u}_1)$ and $\mathcal{B} \models \varphi^{\#}_{p_1,q}(\bar{u}_1, \bar{u}')$, and there is a path of length l for $\varphi^{\#}_{p_1,q}(\bar{u}_1, \bar{u}')$.

Since $\mathcal{B} \models \varphi^{\#}_{p,q'}(\bar{u}, \bar{u}'')$, there also exists $p'_1 \in Q$ and $\bar{u}'_1 \in B^k$ such that $\mathcal{B} \models \varphi_{p,p'_1}(\bar{u}, \bar{u}'_1)$ and $\mathcal{B} \models \varphi^{\#}_{p'_1,q'}(\bar{u}'_1, \bar{u}'')$, and there is a path of length l' for $\varphi^{\#}_{p'_1,q'}(\bar{u}'_1, \bar{u}'')$. From the determinism of Φ, it now follows that $p'_1 = p_1$ and $\bar{u}'_1 = \bar{u}_1$.

Now $\mathcal{B} \models \varphi^{\#}_{p_1,q}(\bar{u}_1, \bar{u}')$ and $\mathcal{B} \models \varphi^{\#}_{p_1,q'}(\bar{u}_1, \bar{u}'')$, so by the induction hypothesis for p_1, $\bar{u}_1$, l, and l', we have that $q' = q$, $\bar{u}'' = \bar{u}'$, and $l' = l$. Thus $n' = n$, and predicates $\varphi^{\#}_{p,q}(\bar{x}, \bar{y})$ and $\varphi^{\#}_{p,q'}(\bar{x}, \bar{y})$ are functional and exclusive.

(ii) Let Φ be a $Q \times Q$ matrix of $\mathrm{FO}(\mathrm{TC}^k)$ formulas $\varphi_{p,q}(\bar{x}, \bar{y})$, where $p, q, \bar{x}, \bar{y}$ are as before. Without loss of generality, we may assume that $Q = \{1, \ldots, m\}$. Let $0 \leq l \leq m$. We define predicates $\varphi^{(l)}_{p,q}(\bar{x}, \bar{y})$ in the same manner as $\varphi^{\#}_{p,q}(\bar{x}, \bar{y})$, but with an additional requirement that the intermediate states $p_1, \ldots, p_{n-1}$ must be chosen from the set $\{1, \ldots, l\}$. We show by induction on l how to construct a matrix $\Phi^{(l)}$ of predicates $\varphi^{(l)}_{p,q}(\bar{x}, \bar{y})$ such that each predicate $\varphi^{(l)}_{p,q}(\bar{x}, \bar{y})$ is an $\mathrm{FO}(\mathrm{TC}^k)$ formula. For $l = m$, all the states in Q are allowed as intermediate states, so we notice that $\Phi^{(m)} = \Phi^{(\#)}$.

For $l = 0$, no intermediate states are allowed, meaning that the length of Φ-path for all predicates $\varphi^{(0)}_{p,q}(\bar{x}, \bar{y})$ is one. So, we let $\Phi^{(0)} = \Phi$, which is in $\mathrm{FO}(\mathrm{TC}^k)$.

Let $\Phi^{(l)}$ be in $\mathrm{FO}(\mathrm{TC}^k)$. We show how to construct $\Phi^{(l+1)}$. Assume that $\mathcal{B}$ is a word model such that $\mathcal{B} \models \varphi^{(l+1)}_{p,q}(\bar{u}, \bar{u}')$, for some $\bar{u}, \bar{u}' \in B^k$. Then for some $n \geq 1$, there exists a Φ-path of length n for $\varphi^{(l+1)}_{p,q}(\bar{u}, \bar{u}')$. If the path is such that $p_i \neq l+1$, for all $0 < i < n$, then we have $\mathcal{B} \models \varphi^{(l)}_{p,q}(\bar{u}, \bar{u}')$. In that case, we already have (from the construction of $\Phi^{(l)}$) an $\mathrm{FO}(\mathrm{TC}^k)$ formula $\varphi^{(l)}_{p,q}(\bar{x}, \bar{y})$.

If the path is such that $p_i = l+1$ for some $0 < i < n$, then there exist indices i_0 and i_1, such that $0 < i_0 \leq i_1 < n$, and i_0 is the first and i_1 is the last index for which $p_{i_0} = p_{i_1} = l+1$. If $i_0 \neq i_1$, there may be indices i, such that $i_0 < i < i_1$ and $p_i \neq l+1$. In that case, on the path from state p to state q, there is (at least) one loop from state $l+1$ back to itself, before the path continues from state $l+1$ to state q.

The transitive closure of predicate $\varphi^{(l)}_{l+1,l+1}(\bar{x}, \bar{y})$ contains all the pairs of tuples which consist of the first and the last tuple of some path (that possibly contains several loops) from state $l+1$ back to itself. Note that the transitive closure is applied to a predicate from $\Phi^{(l)}$, so the states (other than $l+1$) visited on these paths are among the states $1, \ldots, l$ as wanted.

Since the states $p_1, \ldots, p_{i_0-1}, p_{i_1+1}, \ldots, p_{n-1}$ are also among $1, \ldots, l$, it follows from $\mathcal{B} \models \varphi^{(l+1)}_{p,q}(\bar{u}, \bar{u}')$, that there exist intermediate k-tuples $\bar{u}_{i_0}, \bar{u}_{i_1} \in B^k$ such that: $\mathcal{B} \models \varphi^{(l)}_{p,l+1}(\bar{u}, \bar{u}_{i_0})$ and $\mathcal{B} \models \varphi^{(l)}_{l+1,q}(\bar{u}_{i_1}, \bar{u})$. Now we have that

$$\mathcal{B} \models \varphi^{(l)}_{p,l+1}(\bar{u}, \bar{u}_{i_0}) \wedge \Big(\bar{u}_{i_0} = \bar{u}_{i_1} \vee [\mathrm{TC}_{\bar{x},\bar{y}}\, \varphi^{(l)}_{l+1,l+1}(\bar{x}, \bar{y})](\bar{u}_{i_0}, \bar{u}_{i_1})\Big) \wedge \varphi^{(l)}_{l+1,q}(\bar{u}_{i_1}, \bar{u}').$$

From the construction of $\Phi^{(l)}$ we have that the predicates $\varphi^{(l)}_{p,l+1}(\bar{x}, \bar{y})$, $\varphi^{(l)}_{l+1,l+1}(\bar{x}, \bar{y})$, and $\varphi^{(l)}_{l+1,q}(\bar{x}, \bar{y})$ are $\mathrm{FO}(\mathrm{TC}^k)$ formulas. Then the formula

$$\varphi^{(l)}_{p,q}(\bar{x}, \bar{y}) \vee \exists \bar{z} \exists \bar{z}' \Big(\varphi^{(l)}_{p,l+1}(\bar{x}, \bar{z}) \wedge \Big(\bar{z} = \bar{z}' \vee [\mathrm{TC}_{\bar{x},\bar{y}}\, \varphi^{(l)}_{l+1,l+1}(\bar{x}, \bar{y})](\bar{z}, \bar{z}')\Big) \wedge \varphi^{(l)}_{l+1,q}(\bar{z}', \bar{y})\Big)$$

for predicate $\varphi^{(l+1)}_{p,q}(\bar{x}, \bar{y})$ is in $\mathrm{FO}(\mathrm{TC}^k)$.

(iii) This part can be proved in a similar way as the previous part (ii), but there are some details we have to consider. We want to take the deterministic transitive closure of predicate $\varphi^{(l)}_{l+1,l+1}(\bar{x}, \bar{y})$ to get the needed $\mathrm{FO}(\mathrm{DTC}^k)$ formula for predicate $\varphi^{(l+1)}_{p,q}(\bar{x}, \bar{y})$. If there are tuples $\bar{u}_j, \bar{u}_{j+1}, \bar{u}'_{j+1} \in B^k$ such that $\bar{u}_{j+1} \neq \bar{u}'_{j+1}$, but both $\mathcal{B} \models \varphi^{(l)}_{l+1,l+1}(\bar{u}_j, \bar{u}_{j+1})$, and $\mathcal{B} \models \varphi^{(l)}_{l+1,l+1}(\bar{u}_j, \bar{u}'_{j+1})$ hold, then the transitive closure cannot always be changed into a deterministic one. If the path for $\mathcal{B} \models [\mathrm{TC}_{\bar{x},\bar{y}}\, \varphi^{(l)}_{l+1,l+1}(\bar{x}, \bar{y})](\bar{u}_{i_0}, \bar{u}_{i_1})$ goes through such a tuple $\bar{u}_j$, the deterministic version of the transitive closure does not contain the pair $(\bar{u}_{i_0}, \bar{u}_{i_1})$, and the $\mathrm{FO}(\mathrm{DTC}^k)$ formula for $\varphi^{(l+1)}_{p,q}(\bar{x}, \bar{y})$ is not true when it should be.

However, the determinism of Φ ensures that each predicate of the form $\varphi^{(l)}_{r,l+1}(\bar{x},\bar{y})$ is functional, making the situation described above impossible to occur. The functionality of each $\varphi^{(l)}_{r,l+1}(\bar{x},\bar{y})$ can be shown by induction similarly to the proof of part (i). The only difference is that unlike state q in part (i), state $l+1$ does not need to be final. This is the case since any Φ-path for $\varphi^{(l)}_{r,l+1}(\bar{x},\bar{y})$, by definition of $\Phi^{(l)}$, cannot contain $l+1$ as an intermediate state, and thus paths to state $l+1$ cannot be extended.

5.2 Formulas Describing Steps of the Automaton

In the following, we use notations min and S for the minimal element of the domain and the successor relation, respectively (see examples 2.12 and 2.13).

Lemma 5.4. *Let $k \geq 1$ and L be a language accepted by some* $\mathrm{D2PA}^k$*. Then L is definable in* $\mathrm{FO}(\mathrm{DTC}^k)$*.*

Proof. Let M be a $\mathrm{D2PA}^k$ that accepts language L. Without loss of generality, we may assume that $M := (Q, \Sigma, X, q_0, A, I)$ is such that

(1) accepting states do not have outgoing instructions:
if $p, q \in Q$ and $\langle p, \chi, q\rangle \in I$, then $p \notin A$,

(2) the initial state is not accepting: $q_0 \notin A$, and

(3) computations between dropping and retrieving a pebble are nonempty:
if $\langle p, \mathrm{drop}_i(x), q\rangle \in I$, then $\langle q, \mathrm{retrieve}(x), r\rangle \notin I$.

The first assumption ensures that the automaton halts when it reaches an accepting state. We make the other two assumptions so that the accepting computations and computations between dropping and retrieving a pebble are nonempty, and we can use Lemma 5.3.

Additionally, we assume that the automaton M uses n pebbles, $x_n, \ldots, x_1$ such that it always places pebbles on the tape in the given order, starting with the pebble x_n. Now the automaton M can be divided to $n+1$ different 'levels' according to how many of its pebbles are still available for use (i. e. not already placed on the tape). We view these levels as automata $M_n, \ldots, M_0$, where each automaton M_l has pebbles $x_l, \ldots, x_1$. For the automaton M_l, the pebbles $x_n, \ldots, x_{l+1}$ have fixed positions on the tape, and while M_l can test for their presence, it cannot move them. When the automaton M_l drops the pebble x_l, it can be thought that it then queries the automaton M_{l-1} where to go in the input tape, moves there, and retrieves the pebble x_l from a distance. Note that when any of the pebbles $x_{l-1}, \ldots, x_1$ is dropped during a query, we view it as a pebble dropped by the corresponding automaton, meaning that each pebble x_i is thought to be dropped by the automaton M_i, $0 < i < l$.

We partition the set of states as $Q = Q_n \cup \cdots \cup Q_0$, where each Q_l is the set of states where the pebbles available for dropping are exactly $x_l, \ldots, x_1$. Such partition is possible since the pebbles are dropped in the fixed order $x_n, \ldots, x_1$, and the automaton M keeps them nested. For each $0 \leq l \leq n$, the automaton M_l can be viewed as the restriction of the automaton M to the states in Q_l. Note that $q_0 \in Q_n$ and $A \subseteq Q_n$, but initial and accepting states are not specified for automata M_l, when $l < n$.

We show that computations of the automaton M on input words $w \in \Sigma^+$ can be expressed as an $\mathrm{FO}(\mathrm{DTC}^k)$ formula such that the formula is satisfied in the word model for w if and only if M accepts w. We do this inductively by showing how to express the computations of each M_l, $l \geq 0$.

Let $\mathcal{B}_w$ be the word model for w. For each automaton M_l, we construct a matrix $\Phi^{(l)}$ of predicates $\varphi^{(l)}_{p,q}$, where $p, q \in Q_l$. The predicates $\varphi^{(l)}_{p,q}$ correspond to the single steps of automaton M_l. This means that $\mathcal{B}_w \models \varphi^{(l)\#}_{p,q}(\bar{u}, \bar{u}')$ if and only if M_l has a nonempty computation from configuration $[p, \bar{u}, \sigma]$ to configuration $[q, \bar{u}', \sigma]$, where σ represents the pebble stack $\sigma = (x_n, v_n) \ldots (x_{l+1}, v_{l+1})$. In the matrix $\Phi^{(l)}$, there are additional free variables $x_n, \ldots, x_{l+1}$ for positions $v_n, \ldots, v_{l+1}$ of the pebbles that have already been placed on the tape.

When we have the pebble stack $\sigma = (x_n, v_n) \ldots (x_{l+1}, v_{l+1})$, either $l = 0$ or the pebble x_l exists but is not currently placed on the tape. Let the configurations $[p, \bar{u}, \sigma]$ and $[q, \bar{u}', \sigma]$ be such that $[p, \bar{u}, \sigma] \vdash_{M_l, w} [q, \bar{u}', \sigma]$. We first handle the cases where $l = 0$, or $l \geq 1$ but the automaton M_l does not drop the pebble x_l between these configurations. For each of its heads i, the automaton M_l can move left or right, test the symbol of the current cell, or the presence of any of the pebbles $x_n, \ldots, x_{l+1}$ in the current position of the head i. This means that the relation between the two configurations, the predicate $\varphi^{(l)}_{p,q}(\bar{u}, \bar{u}')$, can be expressed by the following first-order logic formulas corresponding to the instructions of the automaton:

instruction:	formula:
$\langle p, \mathrm{right}_i, q \rangle$	$S(u[i], u'[i]) \wedge \bigwedge_{h \neq i} u[h] = u'[h]$
$\langle p, \mathrm{left}_i, q \rangle$	$S(u'[i], u[i]) \wedge \bigwedge_{h \neq i} u[h] = u'[h]$
$\langle p, \mathrm{symb}_{i,\alpha}, q \rangle$	$P_\alpha(u[i]) \wedge \bigwedge_h u[h] = u'[h]$
$\langle p, \neg\mathrm{symb}_{i,\alpha}, q \rangle$	$\neg P_\alpha(u[i]) \wedge \bigwedge_h u[h] = u'[h]$
$\langle p, \mathrm{peb}_i(x_m), q \rangle$	$u[i] = x_m \wedge \bigwedge_h u[h] = u'[h]$
$\langle p, \neg\mathrm{peb}_i(x_m), q \rangle$	$\neg u[i] = x_m \wedge \bigwedge_h u[h] = u'[h]$,

where S is the successor relation. Note that if M_l was nondeterministic, $\varphi^{(l)}_{p,q}(\bar{u}, \bar{u}')$ would

be, in general, a disjunction of some of the formulas above.

If $l \geq 1$, the automaton M_l may also place the pebble x_l on the input tape. In that case, M_l drops the pebble x_l in state p, simulates the automaton M_{l-1}, retrieves the pebble x_l, and then changes to state q. Then the existence of a computation from configuration $[p, \bar{u}, \sigma]$ to $[q, \bar{u}', \sigma]$ requires that there are instructions $\langle p, \mathrm{drop}_i(x_l), p' \rangle$ and $\langle q', \mathrm{retrieve}(x_l), q \rangle$, such that M_{l-1} has a nonempty computation from configuration $[p', \bar{u}, \sigma']$ to $[q', \bar{u}', \sigma']$, where $\sigma' = \sigma(x_l, u[i])$. The automaton M_l has such a computation from $[p, \bar{u}, \sigma]$ to $[q, \bar{u}', \sigma]$, if and only if

$$\mathcal{B}_w \models \bigvee_{q'} \varphi^{(l-1)\#}_{p',q'}(\bar{u}, \bar{u}'), \tag{5.5}$$

where the disjunction is taken over all q' for which there exists instruction $\langle q', \mathrm{retrieve}(x_l), q \rangle$, and the free variable x_l is replaced with $u[i]$, i.e the position at which M_l placed the pebble x_l in state p. Thus the predicate $\varphi^{(l)}_{p,q}(\bar{u}, \bar{u}')$ can be expressed by the formula $\bigvee_{q'} \varphi^{(l-1)\#}_{p',q'}(\bar{u}, \bar{u}')$. Note that due to the determinism of M, states q' are final in $\Phi^{(l-1)\#}$: since $q \notin Q_{l-1}$ and every q' is such that there is an instruction $\langle q', \mathrm{retrieve}(x_l), q \rangle$, the automaton M_{l-1} does not have outgoing instructions from state q'.

All the other predicates $\varphi^{(l)}_{p,q}$ do not correspond to any steps of M_l, so they are defined to be false. Now we have constructed a step matrix $\Phi^{(l)}$. By the induction hypothesis, the matrix $\Phi^{(l-1)}$ is deterministic and in $\mathrm{FO}(\mathrm{DTC}^k)$, so by parts (i) and (iii) of Lemma 5.3, the matrix $\Phi^{(l-1)\#}$ is semi-deterministic and in $\mathrm{FO}(\mathrm{DTC}^k)$. This means that the predicates $\varphi^{(l-1)\#}_{p',q'}$ are $\mathrm{FO}(\mathrm{DTC}^k)$ formulas, and functional and exclusive for states q' that are final in $\Phi^{(l-1)\#}$. If tuples $\bar{u}, \bar{u}'$, and state p' are such that 5.5 holds, the tuple $\bar{u}'$ is unique, and there is a unique state q' such that $\mathcal{B}_w \models \varphi^{(l-1)\#}_{p',q'}(\bar{u}, \bar{u}')$. Thus for given tuple $\bar{u}$ and state p', 5.5 holds for at most one of the disjuncts, and the predicate $\varphi^{(l)}_{p,q}(\bar{u}, \bar{u}')$ expressed by the disjunction is functional and exclusive.

Since the automaton M is deterministic, so is M_l. From the determinism of M_l, and if $l \geq 1$, from the semi-determinism of $\Phi^{(l-1)\#}$, it follows that $\Phi^{(l)}$ is deterministic. Since $\Phi^{(l)}$ is also in $\mathrm{FO}(\mathrm{DTC}^k)$, from Lemma 5.3 (iii), we have that $\Phi^{(l)\#}$ is in $\mathrm{FO}(\mathrm{DTC}^k)$.

For all $0 \leq l \leq n$, the matrix $\Phi^{(l)\#}$ contains the formulas that describe the possible computations of the automaton M_l. Since the computations of M_n are the same as M's, for every $w \in \Sigma^+$, we have

$$\mathcal{B}_w \models \bigvee_{q \in A} \varphi^{(n)\#}_{q_0,q}(\bar{0}, \bar{0}),$$

if and only if the automaton M has a computation on w starting from the initial state q_0 with all the heads in the position of the first cell on the tape, and ending in some accepting state q, again with all the heads in the position of the first cell (i.e. the automaton M

accepts w). Recall that the element $0 \in \mathrm{Dom}(\mathcal{B}_w)$ corresponds to the position of the first cell on the input tape, the cell which contains the symbol $\triangleright$. In general, the minimal element in a word model is not 0. If min denotes the minimal element, the k-tuple $\bar{0}$ in the word model $\mathcal{B}_w$ corresponds to the k-tuple $\overline{\mathrm{min}}$ in an arbitrary word model for w. Since min is first-order definable, we can substitute $\overline{\mathrm{min}}$ for $\bar{0}$ in the formula above, and obtain a sentence[2] in $\mathrm{FO}(\mathrm{DTC}^k)$ such that it is satisfied in any word model for w if and only if the automaton M accepts w.

Lemma 5.6. *Let $k \geq 1$ and L be a language accepted by some $\mathrm{N2PA}^k$. Then L is definable in $\mathrm{FO}(\mathrm{posTC}^k)$.*

Proof. We notice that in the proof of Lemma 5.4, the determinism of M is only needed for applying parts (i) and (iii) of Lemma 5.3. By omitting the assumption that M is deterministic, we can use part (ii) of Lemma 5.3 to make the proof work also in the nondeterministic case – we only have to ensure that the formulas that we obtain are in $\mathrm{FO}(\mathrm{posTC}^k)$. In the proof of Lemma 5.4, negation is only applied to atomic formulas for the negative tests of the automaton, i.e. to check that there is no specific pebble or symbol in the cell. Additionally, all the first-order definitions needed in the proof, e.g. for the relation S and the minimal element min, can be done with formulas where negation appears only in front of atomic formulas. As we construct the formulas in the proof of Lemma 5.3 (ii) without adding negations, the formulas that we obtain are in $\mathrm{FO}(\mathrm{posTC}^k)$.

[2]To obtain the sentence whose models are the word models for all the words $w \in M(L)$, we also need to take a conjunction with the sentence φ_W that says that the model is a word model.

Chapter 6

Transitive Closure Logic and Automata: the Deterministic and the Nondeterministic Case

In this chapter, we combine the lemmas from the previous two chapters to get the main theorem: deterministic two-way k-head automata with nested pebbles capture first-order logic with k-ary deterministic transitive closure. We also mention a corollary of the theorem and discuss the nondeterministic case where the restriction to $\mathrm{FO}(\mathrm{posTC}^k)$ is needed. The case of singlehead automata and unary transitive closure has its own section.

6.1 Results for $\mathbf{FO(DTC}^k\mathbf{)}$ and $\mathbf{FO(posTC}^k\mathbf{)}$

By combining Lemmas 4.1 & 5.4, and Lemmas 4.2 & 5.6 from the previous two chapters, we obtain the following theorems concerning the relationship between the transitive closure logics and the multihead automata with nested pebbles:

Theorem 6.1. *Let $k \geq 1$ and L be a language. Then L is definable in $\mathrm{FO}(\mathrm{DTC}^k)$ if and only if there exists a $\mathrm{D2PA}^k$ that accepts L.*

Theorem 6.2. *Let $k \geq 1$ and L be a language. Then L is definable in $\mathrm{FO}(\mathrm{posTC}^k)$ if and only if there exists an $\mathrm{N2PA}^k$ that accepts L.*

Recall that in the proof of Lemma 4.1, the deterministic automata are constructed to halt on every input. As observed in [5], then from Theorem 6.1 it follows that:

Corollary 6.3. *Let $k \geq 1$, and let M be a $\mathrm{D2PA}^k$. Then there exists another $\mathrm{D2PA}^k$ M' such that $L(M') = L(M)$ and M' always halts.*

Since the class of languages definable in $\mathrm{FO}(\mathrm{DTC}^k)$ is closed under complement and union, the same holds for the class of languages accepted by automata $\mathrm{D2PA}^k$. Note that closure under complement and union also follow directly from Corollary 6.3. In the nondeterministic case, the union closure property can similarly be transferred from logic to automata. For closure under complement, this cannot be done as the logic we consider in the nondeterministic case is $\mathrm{FO}(\mathrm{posTC}^k)$.

If the class of languages accepted by nondeterministic automata $\mathrm{N2PA}^k$ was closed under complement, there would exist corresponding automata also for subformulas of the form $\neg\psi_1$, where $\psi_1 \in \mathrm{FO}(\mathrm{TC}^k)$ (cf. the induction proof for Lemma 4.2). This would mean that there exists an automaton for any language definable in $\mathrm{FO}(\mathrm{TC}^k)$, and the restriction to $\mathrm{FO}(\mathrm{posTC}^k)$ is not necessary in Theorem 6.2. But as it is not known whether the class is closed under complement in the case of nondeterministic automata $\mathrm{N2PA}^k$, we do not know if the restriction to $\mathrm{FO}(\mathrm{posTC}^k)$ can be removed from the theorem.

On the other hand, it is known (see e.g. [4]) that on ordered structures:

$$\mathrm{FO}(\mathrm{posTC}) \equiv \mathrm{FO}(\mathrm{TC}).$$

Each formula $\varphi \in \mathrm{FO}(\mathrm{TC}^k)$ is clearly in $\mathrm{FO}(\mathrm{TC})$, but since the known translation to $\mathrm{FO}(\mathrm{posTC})$ is such that the equivalent translated formula has occurrences of higher-arity transitive closure than the original formula, the translated formula is not in $\mathrm{FO}(\mathrm{posTC}^k)$ anymore.

This is not the case for $\mathrm{FO}(\mathrm{DTC}^k)$. On finite structures, logics $\mathrm{FO}(\mathrm{posDTC})$ and $\mathrm{FO}(\mathrm{DTC})$ are known to be expressively equivalent (see e.g. [4]), and since the translation is such that it preserves the arities of the transitive closures appearing in the formula, we also know that:

$$\mathrm{FO}(\mathrm{posDTC}^k) \equiv \mathrm{FO}(\mathrm{DTC}^k)$$

on finite structures. This is also in line with the fact that in the proof of Lemma 5.4, formulas for deterministic automata could be defined such that they are in $\mathrm{FO}(\mathrm{posDTC}^k)$.

6.2 Singlehead Automata

In the case of the singlehead automata, some of the considerations in the previous section become irrelevant. As mentioned earlier at the end of Chapter 2, the languages definable in $\mathrm{FO}(\mathrm{DTC}^1)$ over strings are exactly the regular languages, and this is also the case for the logic $\mathrm{FO}(\mathrm{TC}^1)$. Knowing that, we can show that for singlehead pebble automata and first-order logic with transitive closure of arity one, there is no difference in the deterministic and nondeterministic case in the sense of the following theorem:

Theorem 6.4. *For a language L, the following are equivalent:*

(1) *L is regular*

(2) *L is definable in* $\mathrm{FO}(\mathrm{DTC}^1)$

(3) *L is definable in* $\mathrm{FO}(\mathrm{TC}^1)$

(4) *L is accepted by a* $\mathrm{D2PA}^1$

(5) *L is accepted by an* $\mathrm{N2PA}^1$

Proof. The equivalence holds by the following chain of implications:

$$(1) \implies (2) \implies (4) \implies (5) \implies (3) \implies (1).$$

The implications (1) $\implies$ (2) and (3) $\implies$ (1) follow from the fact mentioned at the beginning of this section, see [1] for the proof. From Lemmas 4.1 and 5.6, we obtain that (2) $\implies$ (4) and (5) $\implies$ (3), respectively. The implication (4) $\implies$ (5) is clear, since every deterministic automaton is also a nondeterministic one.

Since the regular languages can also be characterized as the languages accepted by deterministic one-way singlehead automata without any pebbles, we also see that in the case of singlehead automata on strings, neither the nested pebbles nor the ability to move both ways increase the expressive power of the automata.

Theorem 6.4 cannot be extended to multiple heads and higher arities by using a similar proof. As observed in [1] (in the case of automata without pebbles), the proof of the theorem relies on the fact that the *Büchi-Elgot-Trakhtenbrot theorem* (see e.g. Theorem 6.2.3 in [4]) can be used to show that the implication (3) $\implies$ (1) holds. The Büchi-Elgot-Trakhtenbrot theorem states that language L is regular if and only if it is definable in monadic second-order logic MSO. Transitive closure can be expressed in MSO: let $\varphi = [\mathrm{TC}_{x,y}\, \psi]st$ be in $\mathrm{FO}(\mathrm{TC}^1)$, and define an MSO formula[1]

$$\begin{aligned}\theta(s) :=& A(s) \wedge \forall x \forall y((A(x) \wedge \psi(x,y)) \to A(y)) \wedge \\ & (\forall B(B(s)) \wedge (\forall x \forall y((B(x) \wedge \psi(x,y)) \to B(y)) \to \\ & \forall z(A(z) \to B(z))).\end{aligned}$$

As the formula $\theta(s)$ says that the set A is the minimal set that contains s and is closed under ψ, the MSO formula $\varphi' := \exists A(\theta(s) \wedge A(t))$ is as wanted. Now it can be seen that to express k-ary transitive closure in a similar manner, one would need to quantify over k-ary relations, resulting in that the formula φ' would not be in MSO. Without the restriction of second-order quantification to quantification over sets, the expressive power of second-order logic SO corresponds to the polynomial hierarchy PH.

[1] Here all the subformulas of the form $\neg\varphi \vee \psi$ have been expressed as $\varphi \to \psi$ for the sake of clarity.

Chapter 7

Related Results and Questions

In previous chapters, we have considered some results concerning the fragments $\mathrm{FO}(\mathrm{DTC}^k)$ and $\mathrm{FO}(\mathrm{posTC}^k)$ of transitive closure logics. Since a correspondence between k-head automata and k-ary transitive closure logics has been established, it can be used for transferring results between automata theory and finite model theory. For example, there exist model theoretical tools such as Ehrenfeucht-Fraïssé games for transitive closure logics (see [3] and Chapter 8 of [4]). Such games for transitive closure logics are simpler for formulas with lower arity transitive closures, so properties of simpler automata (with a small number of heads) can be expressed as properties of these simpler formulas, and then be studied with these games. Naturally, the same applies also vice versa: the fragments $\mathrm{FO}(\mathrm{DTC}^k)$ and $\mathrm{FO}(\mathrm{posTC}^k)$ can be studied by investigating properties of the corresponding k-head automata.

Both multihead automata and transitive closure logics have connections to the space complexity classes $\mathrm{DSPACE}(\log n)$ and $\mathrm{NSPACE}(\log n)$, which are, respectively, the classes of languages decidable by deterministic and nondeterministic Turing machines using a logarithmic amount of memory space. In this chapter, we use the usual denotations $\mathrm{L} = \mathrm{DSPACE}(\log n)$ and $\mathrm{NL} = \mathrm{NSPACE}(\log n)$ for these two complexity classes.

One of the main results in the field of descriptive complexity is that the complexity classes L and NL are characterized by logics FO(DTC) and FO(TC), respectively [5, 10]. On the other hand, these classes can also be characterized by two-way multihead automata: if any finite number of heads is allowed, the languages accepted by deterministic two-way multihead automata are exactly the languages in the complexity class L. The analogous characterization holds for the nondeterministic two-way multihead automata and the class NL. (See e.g. [1, 5, 9].) In the light of Lemmas 5.4 and 5.6, we can see that in the latter two characterizations, it does not make any difference whether the automata have nested pebbles or not.

It is known that the class NL is closed under complement, so the class of languages

accepted by some $\mathrm{N2PA}^k$ for any $k \geq 1$ (i.e. any finite number of heads is allowed) is also closed under complement. It is not known whether this is the case for the class of languages accepted by automata $\mathrm{N2PA}^k$ for a fixed k (see also Section 6.1).

It is an open problem whether these two complexity classes are the same, i.e. whether $\mathrm{L} = \mathrm{NL}$. By the correspondences between the space complexity classes and transitive closure logics discussed above, we have the following well-known result (see e.g. [4]):

Theorem 7.1. $\mathrm{L} = \mathrm{NL}$ *if and only if* $\mathrm{FO(DTC)} \equiv \mathrm{FO(TC)}$ *on ordered structures.*

For the two-way multihead automata, it is also known (see [1, 7]) that:

Theorem 7.2. $\mathrm{L} = \mathrm{NL}$ *if and only if every language accepted by some nondeterministic two-way three-head finite automaton is accepted by some deterministic two-way multi-head finite automaton.*

In fact, the above result has been improved in [13] by showing that the relation remains valid even for nondeterministic one-way two-head automata:

Theorem 7.3. $\mathrm{L} = \mathrm{NL}$ *if and only if every language accepted by some nondeterministic one-way two-head finite automaton is accepted by some deterministic two-way multi-head finite automaton.*

Every formula in $\mathrm{FO(posTC}^2\mathrm{)}$ is clearly in $\mathrm{FO(TC)}$. When we combine this observation with Theorem 7.1, we see that $\mathrm{L} = \mathrm{NL}$ would imply that $\mathrm{FO(posTC}^2\mathrm{)} \leq \mathrm{FO(DTC)}$ on ordered structures. As mentioned already in Section 3.1, the languages accepted by deterministic two-way multihead automata are exactly the languages definable in logic $\mathrm{FO(DTC)}$. Every language accepted by some nondeterministic one-way two-head automaton can also be accepted by some $\mathrm{N2PA}^2$. From Lemma 5.6, we know that every language accepted by some $\mathrm{N2PA}^2$ is definable in $\mathrm{FO(posTC}^2\mathrm{)}$. Thus we obtain the following result (cf. [1]):

Theorem 7.4. $\mathrm{L} = \mathrm{NL}$ *if and only if* $\mathrm{FO(posTC}^2\mathrm{)} \leq \mathrm{FO(DTC)}$ *on ordered structures.*

We see that if $\mathrm{L} \neq \mathrm{NL}$, then there exists a language that is definable in $\mathrm{FO(posTC}^2\mathrm{)}$ but not in $\mathrm{FO(DTC}^k\mathrm{)}$ for any $k \geq 1$. Since $\mathrm{FO(DTC}^k\mathrm{)} \leq \mathrm{FO(posTC}^k\mathrm{)}$ for all $k \geq 1$, this would imply that, on ordered structures, $\mathrm{FO(DTC}^k\mathrm{)} < \mathrm{FO(posTC}^k\mathrm{)}$ for all $k \geq 2$. In other words, if $\mathrm{L} \neq \mathrm{NL}$, starting already from the binary case $k = 2$, the (positive) nondeterministic k-ary transitive closure has more expressive power on ordered structures than the deterministic k-ary transitive closure. Recall that $\mathrm{FO(DTC}^1\mathrm{)} \equiv \mathrm{FO(posTC}^1\mathrm{)}$ on strings.

In addition to comparing the expressive power of deterministic and nondeterministic transitive closure, we can also compare the expressivity of these k-ary fragments in terms

of k. It has been shown in [6] that the arity hierarchies of FO(DTC) and FO(TC) are strict on the class of finite graphs i.e.

$$\mathrm{FO}(\mathrm{DTC}^k) < \mathrm{FO}(\mathrm{DTC}^{k+1}) \text{ and } \mathrm{FO}(\mathrm{TC}^k) < \mathrm{FO}(\mathrm{TC}^{k+1}),$$

for all $k \geq 1$. On ordered structures, this kind of comparison has turned out to be difficult. As mentioned briefly at the end of Section 2.2, it is still open whether the arity hierarchies of FO(DTC) and FO(TC) are strict on ordered structures. As seen in the aforementioned section, in the case $k = 1$ this holds, so the first steps of the hierarchies are strict. Naturally, the corresponding problem for two-way multihead automata with nested pebbles is also open: we do not know if there exist strict hierarchies in L and NL concerning the number of the heads.

Bibliography

[1] Yaniv Bargury and Johann Makowsky. The expressive power of transitive closure and 2-way multihead automata. In Egon Börger, Gerhard Jäger, Hans Kleine Büning, and Michael M. Richter, editors, *Computer Science Logic*, pages 1–14, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[2] Mikołaj Bojańczyk, Mathias Samuelides, Thomas Schwentick, and Luc Segoufin. Expressive power of pebble automata. In *Proceedings of the 33rd International Conference on Automata, Languages and Programming - Volume Part I*, ICALP'06, pages 157–168, Berlin, Heidelberg, 2006. Springer-Verlag.

[3] A. Calò and J. A. Makowsky. The Ehrenfeucht-Fraïssé games for transitive closure. In Anil Nerode and Mikhail Taitslin, editors, *Logical Foundations of Computer Science — Tver '92*, pages 57–68, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[4] H.D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer Berlin Heidelberg, 1999.

[5] Joost Engelfriet and Hendrik Jan Hoogeboom. Automata with nested pebbles capture first-order logic with transitive closure. *CoRR*, abs/cs/0703079, 2007.

[6] Martin Grohe. Arity hierarchies. *Annals of Pure and Applied Logic*, 82(2):103 – 163, 1996.

[7] Markus Holzer, Martin Kutrib, and Andreas Malcher. Multi-head finite automata: Characterizations, concepts and open problems. *Electronic Proceedings in Theoretical Computer Science*, 1:93–107, Jun 2009.

[8] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computability*. Addison-Wesley Longman Publishing Co., Inc., USA, 2nd edition, 2000.

[9] Oscar H. Ibarra. Characterizations of some tape and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata. *J. Comput. Syst. Sci.*, 5:88–117, 1971.

[10] Neil Immerman. Languages that capture complexity classes. *SIAM Journal of Computing*, 16:760–778, 1987.

[11] S. C. Kleene. Representation of events in nerve nets and finite automata. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, NJ, 1956.

[12] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, April 1959.

[13] I.H. Sudborough. On tape-bounded complexity classes and multihead finite automata. *Journal of Computer and System Sciences*, 10(1):62 – 76, 1975.